SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

AD-A207 421

1b. RESTRICTIVE MARKINGS

3 DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution unlimited.

4. PERFORMING ... S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ARO 24621.1-EG-UIE

6a. NAME OF PERFORMING ORGANIZATION	6b OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
Massachusetts Institute of Technology, Civil Engineering	CCRE/PACT	U. S. Army Research Office

6c. ADDRESS (City, State, and ZIP Code)	7b. ADDRESS (City, State, and ZIP Code)
77 Massachusetts Avenue, Rm 1-175 Cambridge, MA 02139	P. O. Box 12211 Research Triangle Park, NC 27709-2211

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
U. S. Army Research Office		DAAL03-86-G-0197

8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS			
	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
P. O. Box 12211 Research Triangle Park, NC 27709-2211				

11. TITLE (Include Security Classification)
DICE: An Object Oriented Programming Environment for Cooperative Engineering Design

12 PERSONAL AUTHOR(S)
Sriram, Duvvuru; Logcher, Robert D.; Groleau, Nicholas; Cherneff, Jonathan

13a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 1/87 TO 1/89	March 20, 1989	51

16. SUPPLEMENTARY NOTATION
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

17 COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Knowledge-Based Expert Systems; Object Oriented Programming; Blackboard Control Techniques; Engineering Design Process; Computer Aided Design

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The development and testing of knowledge based computer tools for the integration and coordination of various phases and participants of the engineering process are described. A system architecture - DICE - is presented which is intended to provide cooperation and coordination among multiple designers working in separate engineering disciplines, using knowledge to estimate interface conditions between disciplines, recording who used any piece of design data created by others, and how such data was used, and checking for conflicts among disciplines, manufacturability, and manufacturing cost and schedule impacts of design decisions. The system is being developed using object oriented programming and blackboard control techniques. Current status of DICE, along with examples in the domain of civil engineering are presented.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

DTIC ELECTE MAY 01 1989 S E

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

DICE: An Object Oriented Programming Environment for Cooperative Engineering Design

Final Report Submitted to U.S. Army Research Office
For Work Conducted During Jan. 1987 - Jan. 1989

D. Sriram, R. D. Logcher, N. Groleau, and J. Cherneff

20 March 1989

Intelligent Engineering Systems Laboratory
Department of Civil Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Approved for public release; Distribution unlimited

This research was funded by U.S. Army Research Office under contract/grant number DAAL0386G0197. The views and/or findings contained in this report are those of the authors and should not be construed as an official department of the Army position policy, or decision, unless so designated by other documentation.

Table of Contents

List of Figures

List of Tables

DICE: An Object Oriented Programming Environment for Cooperative Engineering Design

Abstract

The development and testing of knowledge based computer tools for the integration and coordination of various phases and participants of the engineering process are described. A system architecture - DICE - is presented which is intended to provide cooperation and coordination among multiple designers working in separate engineering disciplines, using knowledge to estimate interface conditions between disciplines, recording who used any piece of design data created by others, and how such data was used, and checking for conflicts among disciplines, manufacturability, and manufacturing cost and schedule impacts of design decisions. The system is being developed using object oriented programming and blackboard control techniques. Current status of DICE, along with examples in the domain of civil engineering are presented.

1 Introduction

On July 17, 1981, two skywalks in the lobby of the Hyatt Regency Hotel in Kansas City collapsed. It was cited as the "most devastating structural collapse ever to take place in the United States"; 114 people died and 186 were injured [24]. This was not only a failure of a physical structural system, but also a failure of the process by which most projects in the U. S. are designed and built. The objective our current research is to provide computer based tools which would help during design and construction to avoid errors of the type made in Kansas City.

The Hyatt failure was attributed to a combination of three events. First, in progressing from the preliminary to detailed design, where joint and connection detailing occurs, the design of the hanger to spandrel beam connection was inadequate. Second, in developing shop drawings, the connection detail was changed by the steel fabricator, thereby "compounding an already critical condition." Third, this second error was not caught during approval checking of the shop drawings by the structural engineers. These were all errors of communication and coordination in the design process, errors caused by the structure of the process, lack of tools used in this process, and focus on documenting the product of design while neglecting "process" and "intent" documentation. These problems also exist in other engineering application areas.

Large engineering projects involve a large number of components and the interaction of multiple technologies. The components included in the product are decided in an iterative design process. In each iteration, interfaces and interface conditions among these components are designed with slack to account for potential variations created when the components and interface values become better known. Iteration proceeds towards increasing detail; design personnel may change, and their numbers expand with increasing level of detail.

The problems facing the engineering industry in the U. S. will be highlighted by considering the design and construction[1] of structures. On a single project, interacting design technologies often come from separate firms or functional groups within a firm, and there is little coordination between designers and contractor(s) during design. Because designers find coordination among themselves difficult, they leave this task to construction managers or the contractor. Thus, working drawings, used to inform the contractor of the product, lack detail. Shop or fabrication drawings are required from the contractor to document details, but potential conflicts among trades are often unrecognized until construction begins. Several undesirable effects are caused by this lack of coordination.

---

[1] Manufacturing in the civil engineering industry is known as *construction*. There are several differences between the construction industry and the manufacturing industry. For example, in manufacturing several hundreds of a single type of product are produced, whereas construction involves the production of one-of-a-kind products. However, the overall engineering process is similar. In this paper the terms manufacturing and construction will be used to denote the *realization or creation of a designed artifact*.

1. The construction process is slowed, work stops when a conflict is found.
2. Prefabrication opportunities are limited, because details must remain flexible.
3. Opportunities for automation are limited, because capital intensive high speed equipment is incompatible with work interruptions from field recognized conflicts.
4. Rework is rampant, because field recognized conflicts often require design and field changes.
5. Conservatism prevades design, because designers provide excessive slack in component interfaces to avoid conflict.
6. The industry is unprepared for the advent of automated construction, as the need for experience in design limits choice to available materials placed by hand.

All of these problems decrease productivity. In addition, failures, such as the Hyatt collapse, occur more often then they should. Overcoming these problems requires significant changes to the design process, together with superior *computer integrated design and construction/manufacturing* (CIDCAM) tools. Those tools must be tailored to the needs of designers who are [2]:

"constantly engaged in searching out various consequences of design decisions [especially those made by others]"

This paper details the development of a prototype system to test new concepts for computer tools to integrate various stages involved in the engineering of a product. The major objectives of this system are to:

1. Facilitate effective coordination and communication in various disciplines involved in engineering.
2. Capture the process by which individual designers make decisions, that is, what information was used, how it was used and what did it create.
3. Forecast the impact of design decisions on manufacturing or construction.
4. Provide designers interactively with detailed manufacturing process or construction planning.
5. Develop intelligent interfaces for automation.

Computer aided tools, which will be collectively called DICE (Distributed and Integrated environment for Computer-aided Engineering), are being currently developed to address the above objectives. DICE will significantly improve productivity by[2]:

- reducing error in design;
- providing more detailed design;
- providing better manufacturing or construction planning;
- allowing easier recognition of design and manufacturing (construction) problems;
- using manufacturability criteria throughout design; and
- advancing automation.

Lessons from the Hyatt failure show that such tools are required. Had the connection designer had access to the concepts of load transmission underlying the preliminary design, local buckling might have been recognized and the joint details changed. Had the fabricator preparing the shop drawings had access to that information, he would have seen that his change violated the purpose of the connection scheme. Had the shop drawing checker seen all these changes together with their intent, he would have recognized the faults in the design.

The engineering design process and problems associated with this process are described in Section 2. Using this

[2]Engineers from several industries that we visited felt that computer aided tools for cooperation and coordination can greatly increase their productivity.

background, an overview of DICE is given. Background material on computer-based technologies used in this work and systems relevant to the current work are presented in Sections 3. A system architecture which utilizes concepts from knowledge-based systems and database management systems is described in Section 4. This is followed by a description of the current status of DICE.

2 Scope of Work

The problems that engineers normally solve fall along *the derivation-formation* spectrum [1]. In derivation-type problems, solutions consist of identifying an outcome or hypothesis from a finite set of outcomes known to the problem solver. By contrast, in formation-type problems, the problem solver has only the knowledge of how to form the solution. A variety of problem solving techniques are invoked to arrive at a solution.

Design and manufacturing (or construction planning) problems fall at the formation end of this spectrum. Design and manufacturing are accomplished by a team of engineers, each knowledgeable in a particular aspect of the problem, but with little knowledge of the decision processes of others. Each could be considered as one of many sources of knowledge, and hence, design and manufacturing (construction) could be viewed as *a process of constructing an artifact which satisfies constraints from many sources by using knowledge which also comes from many sources*. The extent of interactions can be seen by looking at the diverse set tasks, listed in Table 1, that must be performed by a diverse set of professionals during the design, for example, for a high rise building [30].

Planning	Architectural design
Spatial layout	Site planning
Preliminary structural design	Analysis modeling
Component design	Geometric modeling
Substructure design	Cost estimating
Electrical distribution design	Elect. distribution analysis
Mechanical design	Mechanical analysis
HVAC design	HVAC analysis
Vertical transportation design	Regulatory compliance
Various design critics	Fire safety analysis

Table 1: Tasks involved in the Design of a Tall Building

As CAD/CAE becomes more widespread, each of these consultants will be performing increased amount of their work with computer tools, tools which will embody and use their knowledge in their speciality area.

From this view, the stages of design and construction might be described as[3]:

1. **Problem Identification.** The problem, necessary resources, target technology, etc., are identified at this stage.
2. **Specification Generation.** Design requirements and performance specifications are listed.
3. **Concept Generation.** The selection or synthesis of potential design solutions, such as a structural system, is performed. Several alternative designs may be generated.
4. **Analysis.** The response of the system to external effects is determined by using the appropriate model for the system.
5. **Evaluation.** Solutions generated during the Concept Generation stage are evaluated for consistency with respect to the specifications. If several designs are feasible then (normally) an appropriate evaluation function is used to determine the best possible design to refine further.

[3]Similar stages occur in the manufacturing industry.

6. **Detailed Design.** Various components of the system are refined so that all applicable constraints (or specifications) are satisfied.
7. **Design Review.** The detailed design is checked for global consistency.
8. **Construction.** This involves the preparation of shop drawings, development of detailed construction schedules, actual construction, and construction monitoring.

There may be significant deviations between the properties of components assumed or generated at the Concept Generation stage and those determined at the Detailed Design stage, which would necessitate a reanalysis. The process continues until a satisfactory or optimal design is obtained.

During each stage in this process, representatives from the various interacting disciplines meet and discuss potential interactions between the components they envision designing. They use estimates of space needs, structural, heat, and electrical loads, and other factors to set requirements for their systems based on the needs of others. Experience is used to estimate these interfaces. Explanations on how these estimates were determined is seldom sought, except where they cause conflicts between objectives. When individual designers select components and systems during any stage of design, they use and try to develop solutions which satisfy the interface estimates.

The problem with this process is that individual designers often lack sufficient experience in both estimating their interfaces (assessing their impact on others) and in asking for information needed from others. They assume, instead, situations similar to other designs. Similarly, they seldom think about and may even lack knowledge of constructability or management and control of the construction process. This may lead to incompatible component selection and poor choice of design parameters. For example, the use of wide rooms in low cost housing is incompatible with inexpensive construction techniques. The designer is assumed in this process to have sufficient knowledge of construction techniques, materials, and equipment to make proper decisions. This is seldom the case. Also, since the present design process does not document reasons behind decisions, others cannot easily question decisions or improve designs. DICE's framework was developed to obviate the above problems. A simplified view of DICE is shown in Figure 1, where users within their discipline interact with individual CAD tools and KBS for component design and solution generation. These systems automatically communicate with a global system which provides data and support facilities.

An initial (prototype) version of DICE operates on two SUN workstations. The final system will operate on several interacting computers, which will be high speed workstations with good knowledge representation tools. Knowledge representations for support facilities are required to:

1. Estimate and negotiate interface parameters between stages of design, doing so in an interactive manner, when a designer asks for information (i.e., if a designer asks for information that has not yet been developed, knowledge will be used to estimate values);
2. Keep track of who used design information, when, and whether it was estimated or actual values (so that when values change, the design can remain coordinated);
3. Use coded individual knowledge sources to assist in or automate component design, retaining component information about sources of data used in the design, the algorithms or knowledge used, and inputs on design rationale from the user;
4. Operate numerous background processes to check design choices for interferences, violations of interface assumptions, constructability, and cost and schedule impacts;
5. Allow user input and design alterations from either a commercial CAD system or the knowledge representation workstation; and
6. Inform designers of the impacts of initial designs and changes by others on their design choices.

We extend the simplified version of DICE, shown in Figure 1, to address the above issues in Section 4.

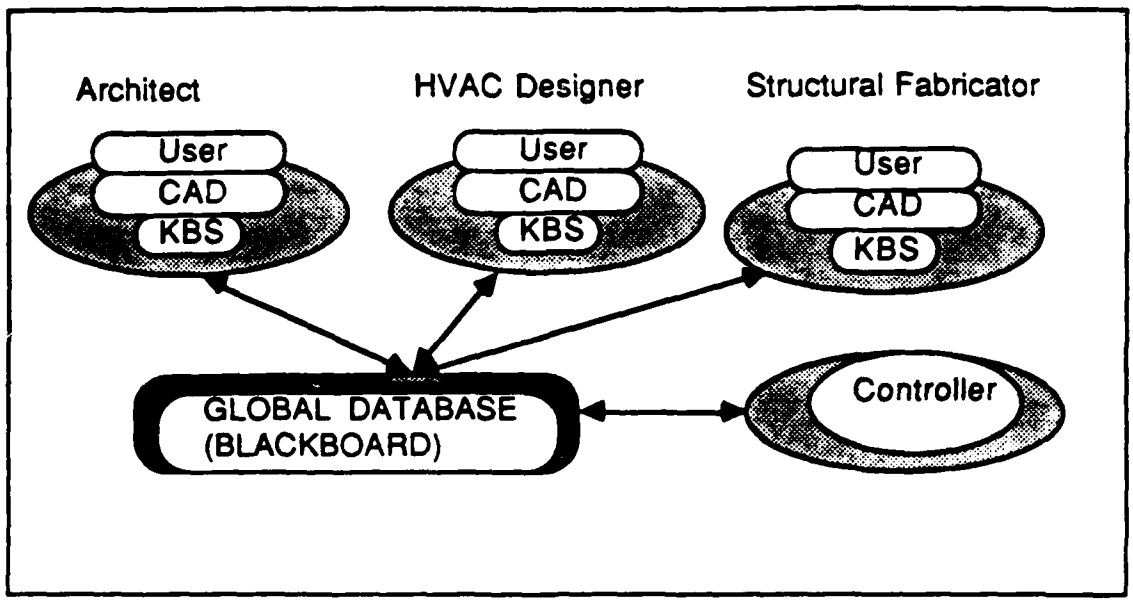


Figure 1: A Simplified View of a Distributed Design Environment

3 Background

There are five technologies are required to realize DICE: Artificial Intelligence (knowledge-based systems, object oriented programming, negotiation theory, etc.), Distributed Databases, Local Area Networks, Deisgn Methodologies (design for assembly, Taguchi's methods, house of quality, hierarchical design models), and Visual Computing (geometric reasoning and user interfaces). In the next section, we will describe some of the computer-based technologies that are utilized in the current DICE prototype. This is followed by a summary of work on negotiation. A review of relevant research work is provided in Section 3.3.

3.1 Relevant Computer-Based Technologies

Developments in computer science and engineering methodologies have provided engineers with a variety of software development tools. The computer-based software development tools that are relevant to this project are:

1. Object Oriented Programming (OOP) Methodologies;
2. Knowledge based systems (KBS);
3. Database management systems (DBMS);
4. Visual computing; and
5. Local area networks.

Object Oriented Programming. Object Oriented Programming (OOP) is a style of programming that involves the use of objects and messages. Objects are defined by Stefik and Bobrow as [37]:

> Objects are entities that combine the properties of procedures and data since they perform computations and save local state.

Objects contain slots and slots may consist of a number of facets. A slot may simply be an attribute or it may be a relation. The facets contain meta-information about the slot.

All actions in object oriented programming are performed through messages. Messages tell the object *what to do* and not *how to do it*. Methods are attached to the object to execute the actions associated with the messages. The

message passing ability in OOP supports the concept of *data abstraction.*

Objects can be grouped into "classes," where each class of objects knows how to perform several actions. Individual instances of objects can be created from a particular class. The Object Oriented programmer builds a system by specifying new classes of objects and their associated methods. Most OOP systems support the concept of "inheritance," where a class of objects may be specified as a "subclass" of another "superclass" of objects. Subclasses and instances inherit methods from their superclass, and are usually more specific entities than their (usually) more general superclass. An object may inherit methods and data from multiple classes through a network of structural relationships. In short, every object has the ability to: *store information, process information, create new information,* and *communicate with other objects. Thus OOP facilitates encoding design and construction knowledge in a disaggregated and modular form.*

As an example consider the following object:

```
BEAM-1
  instance: "Beam"
  M :
  Methods: Display-moment, Calculate-moment
```

The message *(send beam-1 calculate-moment)*, where beam-1 is the object to which the message is addressed, would compute the moment in accordance with the Calculate-moment method. For further details about object oriented programming see [37].

Knowledge-based systems. KBS are computer programs which incorporate knowledge and reason through the application of their knowledge to data about a specific problem. If these systems incorporate human expertise then they are called knowledge-based expert systems (KBES)[4]. A typical KBES consists of three components: *Knowledge-base, Context,* and *Inference Mechanism or Control Mechanism.* Several problem solving architectures used in the Inference Mechanism are described in [36].

In this work, a knowledge-based framework - the Blackboard architecture - that facilitates the integration of diverse sources of knowledge is used [16]. In addition, the work on truth maintenance systems will also be utilized [9, 10].

The Blackboard architecture provides a framework for: 1) integrating knowledge from several sources, and 2) representing multiple levels of problem decomposition. It uses two basic strategies [25]: 1) divide and conquer, and 2) opportunistic problem solving. The divide and conquer strategy is realized by decomposing the context, which is called a *Blackboard,* into several levels depicting the problem solution decomposition, while opportunistic problem solving is achieved by focusing on the parts of the problem that seem promising. The Blackboard architecture has been successfully used in solving a wide range of tasks, such as speech recognition [12], signal processing [26], and planning [16].

Database management systems. Engineers have always dealt with large amounts of data in diverse applications. Hence, storing and manipulating data forms an integral part of the engineering process. Database management systems (DBMS) provide means to store large amounts of data in databases for use by a variety of applications. Data access is controlled through a dictionary so that individual programs need not be changed when the database

[4]For the purpose of this paper, the term KBS and KBES will be used interchangeably.

structure changes. If a problem requires the integration of several geographically distributed databases then we enter the realm of distributed databases. A system that manages these distributed databases is termed as a distributed database management system (DDMS). There are several issues that arise in the development of a DDMS: concurrency control, query processing, reliability, efficiency, etc. (See [3, 7, 27] for further discussion of these issues).

Visual Computing. Engineers make extensive use of diagrams (images) to convey their ideas. They also like to see scientific information (or data) to be conveyed by visual diagrams (images). Hence computer-based systems for engineering should have the ability to: 1) recognize and understand diagrams, and 2) generate diagrams. The study and development of the methodologies required to provide above capabilities in a computer program falls under the realm of *Visual Languages*. Visual languages can be classified into: 1) Visual Information Processing Languages (VIPL), and 2) Visual Programming Languages (VPL) [4]. In VIPL, the objects that are displayed on the screen (by the engineer) have inherent visual meaning, i.e., the object has some semantic meaning associated with it. The task here is to map these objects into their semantic content. An example of VIPL is spatial reasoning about engineering objects. On the other hand in VPL, the visual diagrams are generated on the screen from scientific (or otherwise) data and it is left to the engineer to extract the semantic meaning of these diagrams, e.g., current work on solid modelling. It is also important to realize that these visual languages should be portable. Hence, they should be developed in an environment that is portable across a variety of hardware, such as the X Window system, which is rapidly gaining acceptance as an industry standard.

3.2 Negotiation Theory

Negotiation is a process by which a joint decision is made by two or more parties. The parties first verbalize contradictory demands and then move toward agreement by a process of concession making or search for new alternatives. The parties can range in size and importance from children trying to decide how to divide up a set of toys to nations trying to end a war (which might not be as different as it seems once you really think about it). Irrespective of the size and the type of the parties involved, there seems to be a general body of principles that are applicable to address the negotiation problem. Pruitt's approaches to the negotiation problem has considerable relevancy for DICE [28].

Pruitt's Princliples of Negotiation. Two basic principles of negotiation - the goal/expectation hypothesis and the strategic choice model - are presented in [28].

1. The *goal/expectation hypothesis* states that for most forms of coordinative behavior to be enacted, it is necessary to have both a goal of achieving coordination and some degree of trust in the other party's readiness for coordination.
2. The *strategic choice model* postulates three basic strategies for moving toward agreement: unilateral concession; competitive behavior; and coordinative behavior. These are assumed to be at least partially mutually exclusive. Hence, tradeoffs among them can be expected. Conditions that discourage the use of one strategy should encourage the use of the others and vice versa.

Since negotiation itself can be viewed as a form of coordination, the theory of how coordination develops provides insight into the conditions under which negotiation begins.

Coordination. Coordination occurs when bargainers work together in search of a mutually acceptable agreement. Without the possibility of coordination, negotiation would often take an inordinate amount of time, create much psychological strain, end in disagreement, and/or poison future relations between the participants. Coordination is very common, especially in the later stages of negotiation, when competitive behavior no longer seems very productive. According to Pruitt, two main types of coordination occur in negotiation [28]:

1. *Concession exchange*, in which the parties move toward one another on a single dimension or swap concessions on different dimensions; this is a form of *compromise bargaining*.

2. *Problem-solving discussions*, in which the parties share information about goals and priorities in search of an option that will satisfy both parties' needs, that is, an *integrative agreement*.

For further background on negotiation theory see [15].

3.3 Relevant Systems

The following systems, developed in recent years, address some of the capabilities needed for DICE[5]. However, these systems do not address the problem from a global perspective. Further, full scale implementations of many of these systems do not yet exist.

3.3.1 CAE Systems

The following systems have been developed for computer aided engineering.

The DARPA Initiative in Concurrent Engineering (DICE). The primary goal of DICE2[6] is the development of an advanced design and engineering environment that facilitates rapid prototyping of electro-mechanical parts and high-density electronic assemblies [29]. In DICE2, each workstation is connected to the DICE2 Communication Channel (DCC) through which all information flows, with the help of a Concurrency Manager (CM). Each workstation has a Local Concurrency Manager (LCM). LCM coordinates the flow of Info-Paks (extended NFS-Network File Servers) with the CM to provide transparency. In addition, the workstation also has a local database (LDB) and a set of analysis and support tools, with isolated interfaces. Using, the LCM, LDB and the tools, a domain expert will be able to access remote data, perform computations and communicate the results to others in the network. The LCM will also be used to access External Networks and remote Compute Servers.

Conclusions drawn from local analysis are used to influence design by asserting decision parameters on a Blackboard. A Knowledge Server (KS) is used to retrieve generic information such as that which may be found in a handbook, or transformed information from the Part-Process Organization (POP) database. The simulated Production Facility (SPF) at one end of the DCC is a multi-level simulator that may be used by production engineers, plant designers and others to validate producibility before designs are finalized. The Advanced Prototyping center (APC), at the other end of the DCC, consists of programmable machine tools which may be used to produce actual prototypes prior to production runs at the pilot plant level.

DICE2 is still in a development stage and is yet to be demonstrated on a real world problem. If successfully implemented, this system should provide answers to most of the problems stated here. It is interesting to note that DICE and DICE2 have many similar characteristics.

An Integrated Software Environment for Building Design and Construction (IBDE). Fenves et al. developed an integrated environment - called IBDE (Integrated Building Design Environment) - of processes and information flows for the vertical integration of architectural design, structural design and analysis and construction planning [13]. The integrated environment makes use of a number of AI techniques. The processes are implemented as KBES. A Blackboard architecture is used to coordinate communication between processes. The global information shared among the processes is hierarchically organized in an Object Oriented Programming language.

The Integrated Building Design Environment (IBDE) system is implemented in the form of five vertically integrated Knowledge-Based processes:

1. An architectural planner (ARCHPLAN).

---

[5] In addition, a recent summary of some of the work in computer aided cooperative work appeared in [21].

[6] For the sake of preventing confusion, this system will be referred to as DICE2 and not as DICE.

2. A preliminary structural designer (HI-RISE).
3. A component designer (SPEX).
4. A foundation designer (FOOTER).
5. A construction planner (CONSTRUCTION PLANEX).

The processes communicate with each other in two ways:

1. a message Blackboard is used to communicate project status information such as whether a process is ready to execute, has successfully performed its task or has encountered a failure, and
2. a project database used for storing the information generated and used by the processes

A controller uses the information posted on the Blackboard to initiate the execution of individual processes. The controller also directs the data manager to provide and receive the information shared between the processes. Since the different processes may reside on different machines, the data manager and the Blackboard rely on a local area communication network.

DESTINY. DESTINY is a knowledge-based framework developed for integrating all stages of the structural design process [34]. It consists of several Knowledge Modules (KMs) that communicate through a Blackboard. The Blackboard consists of several levels that define the abstraction hierarchy of objects. The entities generated at various levels in the Blackboard are connected through relational links to form a solution to the structural design problem. The KMs are grouped into: *Strategy*, *Specialist*, and *Resource* KMs. The Specialist KMs perform various tasks of the structural design process. Example KMs are: ALL-RISE, for preliminary structural design; MASON, for structural analysis; DATON, for detailing; and CRITIC for evaluating designs. The Strategy KM controls the design process, while the Resource KMs contain algorithm programs such as Finite Element Analysis.

KADBASE. KADBASE was developed to provide a knowledge-based interface for communication between multiple knowledge-based expert systems and databases [17]. The main components of KADBASE are: 1) The Knowledge-based System Interface (KBSI), which provides the translations (semantic and syntactic) for each KBES for communicating with the Network Data Access Manager (NDAM); 2) Knowledge-Based Database Interface (KBDI), which provides the translations needed for each DBMS for communicating with NDAM; and 3) NADM, which decomposes queries and updates and sends them to the appropriate KBES/DB. A good review of of the work on the use of databases in engineering can be found in [17].

GEMSTONE. GEMSTONE combines the advantages of Object Oriented Programming (OOP) languages with the storage management of traditional DBMS [23]. In addition to the OOP language (called OPAL), GEMSTONE's programming environment also provides an interactive interface: for defining new objects; 2) a windowing package for building user interfaces; and 3) interfaces to conventional programming languages. GEMSTONE was developed for a multi-user environment and has a disk manager for swapping objects to and fro from resident memory into the hard disk.

Katz's Work. Several issues in the management of large design databases, in the realm of VLSI design, are discussed in [18]. The design data management system, described in [18], consists of the following components: 1) Storage component, which manages design data on secondary storage; 2) An Object Oriented database, which supports several semantic relationships between objects of the database; 3) Design Librarian, which coordinates all access to shared design data by making design objects available to various workstations; 4) Recovery subsystem, which manages changes from workstations to database servers; 5) Design Validation subsystem, which assists in determining the validity of an object when changes occur; and 6) Design Transaction manager, which uses the Design Librarian, Recovery subsystem, and the Design Validation subsystem, to ensure that the design objects created are in a consistent state.

Eastman's Work. Eastman and his colleagues have addressed the issue of integrating multiple design databases

in considerable detail [11]. Eastman points out, with an example of piping system design, that a DBMS must be able to handle interactions of several forms, such as: "the loads generated by piping that must be picked up by the structure" and "spatial conflicts between piping and structural systems". He recommends the use of transaction graphs, where nodes denote transactions on a database and links provide the precedence relationships, as a potential solution to the above problem. In addition, he also addresses the problem of concurrent users and communication among multiple users. He proposes that a design database should "have attached methods that other users of the database may be made aware of any assumptions". Although Eastman addressed several important issues in his work, a full scale implementation was hindered by lack of adequate programming methodologies, such as OOP systems and KBS, at that time.

3.3.2 Computer-Aided Negotiation

The following computer-based systems address the negotiation problem.

CALLISTO. The engineering of large complex artifacts involves a number of activities which require the close cooperation of a number of departments. The CALLISTO project emerged out of the realization that classical approaches to project management is inadequate [33]. A prototype, MINI-CALLISTO, was developed to support the needs of an organizational unit by providing means for communication and coordination during the pre-planning stage of the project management process. MINI-CALLISTO has several Knowledge Modules (KMs), each with a similar architecture to MINI-CALLISTO, that communicate through messages. The coordination between various KMs is achieved through a *constraint-negotiation* algorithm, specially designed to address project management problems.

Resources Reallocation Problems. A theory of negotiation to solve resource reallocation problems among multiple agents was developed and tested by Sathi [32]; the resource allocation problem deals with the optimal allocation of resources to agents. In a typical resource allocation situation, there are a set of agents, each with a set of allocated resources working against a set of activities requiring resources. A typical reallocation transaction specifies ownership exchange for one or more resources among agents. A simple transaction involves selling of a resource from one agent to another. A trade involves a two way exchange of resources between two agents. A cascade involves an open or closed chain of buy and sell among more than two agents. In his work, Sathi defines Constraint-Directed Negotiation as a set of qualitative evaluation and relaxation techniques based on human negotiation problem solving. The three constraint relaxation techniques experimented are as follows:

1. *Bridging*: A grouping of buy/sell bids or transactions in order to meet a complex constraint.
2. *Reconfiguration*: A change in the resource attribute value in order to meet the requirements of a buyer.
3. *Logrolling*: Selective constraint violation on less important constraints in order to accept a transaction which is acceptable on more important constraints.

While initial evaluations on constraints were done locally by each agent, the above relaxations were performed during a group problem-solving session, as if performed by a mediator. The automated problem solver uses a mix of local and global knowledge to facilitate cooperative reasoning where some problem solving is done by each agent and some as a group. The individual and group search processes use several aspects of constraints, such as constraint importance and looseness to prioritize the search steps.

Deals Among Rational Agents. A formal framework is presented in [31] that models communication and promises in multi-agent interactions. This framework extends previous work on cooperation without communication and shows the ability of communication to resolve conflicts among agents having disparate goals. Using a deal-making mechanism, agents are able to coordinate and cooperate more easily than in the communication-free model. In addition, there are certain types of interactions where communication facilitates mutually beneficial activity that is otherwise impossible to coordinate.

Negotiation as a Metaphor for Distributed Problem Solving. A framework, called *Contract Net*,that specifies communication and control in a distributed problem solver was developed by Davis and Smith [8]. The kinds of information which must be passed between nodes in order to obtain effective problem-solving behavior is the origin of the negotiation metaphor; task distribution is viewed as a form of contract negotiation. The use of the Contract Net framework is demonstrated in the solution of a simulated problem in area surveillance. The system is based on the assumption that three issues are central to constructing frameworks for distributed problem solving [8]:

1. The fundamental conflict between the complete knowledge needed to ensure coherence and the incomplete knowledge inherent in any distribution of problem solving effort.
2. The need for a problem solving protocol.
3. The utility of negotiation as an organizing principle.

Davis & Smith feel that negotiation is an important aspect of distributed problem solving and consists of three important components: 1) there is a two-way exchange of information, 2) each party to the negotiation evaluates the information from its own perspective, and 3) final agreement is achieved by mutual selection.

Multistage Negotiation in Distributed Planning. Multistage negotiation provides a means by which an agent can acquire enough knowledge to reason about the impact of local activity on non-local state and modify its behavior accordingly. Conry describes a multistage negotiation paradigm for planning in a distributed environment with decentralized control and limited interagent communication [6]. The application domain of interest involves the monitoring and control of a complex communications system. In this domain, planning for service restoral is performed in the context of incomplete and possibly invalid information which may be updated dynamically during the course of planning. In addition, the goal of the planning activity may not be achievable - the problem may be over constrained. Through multistage negotiation, which involves negotiating on primary goals at first and taking secondary goals into account later, a planner is able to recognize when the problem is overcontrained and to find a solution to an acceptable related problem under these conditions. A key element in this process is the ability to detect subgoal interactions in a distributed environment and reason about their impact.

Run-time Conflict Resolution in Cooperative Design. Klein and Lu proposed a conceptual framework for conflict resolution in cooperative design [20]. The basic assumption in this approach is that the different kinds of conflicts that occur in design can be arranged into a set of abstract classes and that conflict resolution strategies can be associated with each conflict class. Klein and Lu define model-based negotiation as the incremental relaxation of constraints derived from one or more KSs involved in a conflict situation. Examples of constraint relaxation include broadening a numerical range or adding members to a constraint. The goal of this relaxation is to produce a satisfiable constraint set while minimizing the decrement in performance from the perspective of the conflicting KSs. This negotiation is called model-based because it is based on a model of how changes in the design lead to changes in performance. The result of such negotiation is a compromise that may be less than optimal locally, but removes the conflict situation so design can continue. To be able to engage in negotiation, each KS must be able to [20]: 1) relate suggested design changes to performance changes; and 2) express the performance changes using a common yardstick so that a reasoned decision can be made.

Negotiation of Conflicts Among Design Experts. Lander and Lesser propose two major types of negotiation operations, with examples from the domain of kitchen design, and describe the design and prototype implementation framework for knowledge-based systems [22]. They identify two types of negotiation based on Pruitt's work:

1. *Compromise bargaining.* It is appropriate when differences in proposed solutions are not too severe, constraints have some built-in flexibility and there are well-defined mechanisms for relaxing or strengthening constraints as necessary.
2. *Integrative bargaining.* When the above conditions don't apply, it may still be possible to come to an agreement by reevaluating the long-term goals and knowledge available. There may be an acceptable solution that is not obvious in the current problem formulation but which could be discovered by

restructuring some underlying assumptions.

Table 2 summarizes the problems addressed, technologies used, and relevancy to the current work.

4 A Framework for A Distributed and Integrated Environment for Computer-Aided Engineering

In Section 1 (page 2) several objectives for a Distributed Integrated environment for Computer-aided Engineering (DICE) have been enumerated. To achieve these goals, a system architecture based on current trends in programming methodologies, object oriented databases, and knowledge based systems is proposed. An overview of DICE is provided in Section 4.1. This is followed by a discussion of various components comprising the system.

4.1 Overview of DICE

DICE can be envisioned as a network of computers and users, where the communication and coordination is achieved, through a global database, by a control mechanism. DICE consists of several Knowledge Modules, a Blackboard, and a Control Mechanism. These terms are clarified below.

1. **Control Mechanism.** The communication, coordination, data transfer, and all other functions define the Control Mechanism; the Control Mechanism could be viewed as an Inference Mechanism.

2. **Blackboard.** The Blackboard is the medium through which all communication takes place. The Blackboard in DICE is divided into three partitions: Coordination, Solution, and Negotiation Blackboards. The Solution Blackboard partition contains the design and construction information generated by various Knowledge Modules, most of which is referred to as the Object Hierarchy containing information about the design product and process, while the Negotiation Blackboard partition consists of the negotiation trace between various engineers taking part in the design and manufacturing (construction) process. The Coordination Blackboard partition contains the information needed for the coordination of various Knowledge Modules.

3. **Knowledge Module.** Each Knowledge Module can be viewed either as: a knowledge based expert system (KBES), developed for solving individual design and construction related tasks, or a CAD tool, such as a database structure, i.e., a specific database, an analysis program, etc., or an user of a computer, or a combination of the above. A KBES could be viewed as an aggregation of Knowledge Sources (KSs). Each KS is an independent chunk of knowledge, represented either as rules or objects. In DICE, the Knowledge Modules are grouped into three categories: Strategy, Specialist, Critic, and Quantitative. The Strategy KMs help the Control Mechanism in the coordination and communication process. The Specialist KMs perform individual specialized tasks of the design and construction process, while the Quantitative KMs are mostly algorithmic CAD tools.

A conceptual view of DICE for design and construction is shown in Figure 2. In it, any of the KMs can make changes or request information from the Blackboard; requests for information are logged with the objects representing the information, and changes to the Blackboard may initiate either of the two actions: finding the implications and notifying various KMs, and entering into a negotiation process, if two or more KMs suggest conflicting changes.

Details of individual components are provided in the following sections.

4.2 Control Mechanism

The Control Mechanism performs two tasks: 1) evaluate and propagate implications of actions taken by a particular KM; and 2) assist in the negotiation process. This control is achieved through the object oriented nature of the Blackboard and a Strategic KM. One of the major and unique differences between DICE and other Blackboard systems is that DICE's Blackboard is more than a static repository of data; DICE's Blackboard is an intelligent database, with objects responding to different types of messages.

System	Purpose	Computer-based Technologies Used	Limitations	Relevancy to Proposed Work	Developer (Date)
DESTINY	An integrated KBS framework for structrual analysis and design	KBS (Blackboards), some DBMS	Only in conceptual stage. Many ideas yet to be tested.	The Blackboard framework. Concept of Strategy Knowledge Module. Symbolic and numeric coupling.	Sriram [CMU/MIT] (1984-)
KADBASE	Provide a KBS interface between multiple KBES and DBMS	KBS (frames) , DBMS	No coordination	Interface definitions between KMs	Howard & Rehak (1986, CMU)
GEMSTONE	Provide an object-oriented DBMS	OOPs, DBMS	No coordination No communication	Disk storage management	Maier, et al. (1986, Oregan)
Katz's Work	Provide DBMS for engineering VLSI design	Objects, DBMS	No coordination	Semantic relationships, storage management, transactions	Katz (1980-, UC Berkley)
Eastman's Work	Integrate multiple project databases	Pascal record structures, DBMS	No concrete implementation	Communications betweens users	Eastman (1981-84, CMU)
CALLISTO	Means for communication and coordination during preplanning stage pf Poject management process.	KBS, with OOPs flavor	Does not address interface issues	Coordination (constraint negotiation)	Sathi, et al. (1986-, CMU/CGI)
IBDE	Integrated building design	Blackboard, Uses DPSK	Simple communication, No negotiation	Same as DESTINY	Fenves, et al (1988, CMU)
DICE2	Cooperative engineering design	Distributed blackboard	Full implementation does not exist	Communication, object oriented database, etc.	Reddy et al. (1988, WVU)
Other Systems (pages 10-11), as described in text					

Table 2: Relevant Systems to Current Work

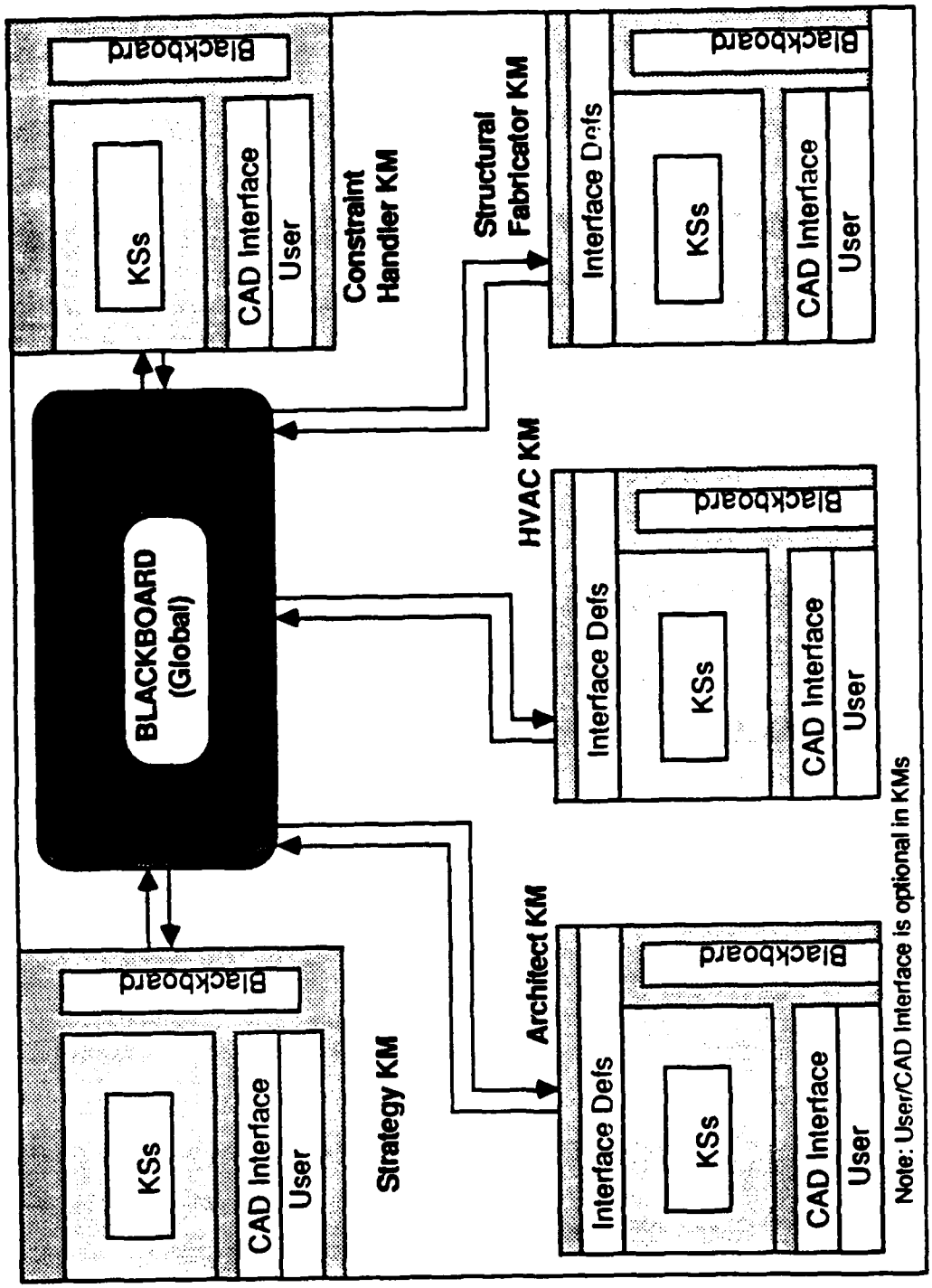


Figure 2: A Conceptual View of DICE for Design and Construction

Task 1 is accomplished through

1. methods associated with objects in the Object-Hierarchy of the Solution Blackboard partition (SBB); and
2. a truth maintenance system which keeps the global database in a consistent state.

If two or more KMs try to access the same object, then the priorities are determined by the Strategy KM and the scheduling information is stored in the Coordination Blackboard partition (CORDBB). A possible trace of events for the Hyatt Regency case is shown in Figure 3, and outlined below.

1. A preliminary design of a building (in the form of objects) which includes loading details and designer's intentions in making certain decisions is posted on to the Solution Blackboard partition by the Conceptual Designer.
2. Let the connection details of a particular joint be represented by the Connection object. The Connection Designer will send a message with details of connections and any assumptions made during the design.
3. The truth maintenance system (TMS) checks to see whether earlier assumptions made by the Conceptual Designer are violated or not.
4. Associated with the Connection object are methods, which indicate the possible KMs that can modify the object. Assume that Fabricator KM is one of them. A message is sent to Fabricator KM to find out whether the connection can be fabricated in the field.
5. Notify the Connection Designer if any problems are anticipated.
6. Sometimes two or more KMs may want to modify or access a particular object in the Solution Blackboard partition. This information is stored in Coordination Blackboard partition and is used by the Control Mechanism.

A possible scenario for task 2 for an interior design problem, which involves the cooperation of an architect and a HVAC engineer, is given below (See Figure 4).

1. Let the Architectural KM post the location and other details of beams in the beam object, whose primary owner is Architectural KM.
2. The HVAC KM would post a design, which makes the assumption that ducts can pass through the beams.
3. Since the object is modified by more than one KM, Solution Blackboard partition checks to see if the object (or objects) being modified has any interaction (interface) constraints. If so then appropriate constraints are stored in the Negotiation Blackboard partition.
4. Solution Blackboard partition, then, sends a message to Strategic KM to check the constraints.
5. Strategic KM sends a message to the Constraint Handling KM (CHKM). CHKM checks to see if the interaction (interface) constraints are satisfied. If so, a message is sent to the Solution Blackboard partition and appropriate actions are taken (step 6).
6. If the interaction constraints are not satisfied then the Strategic KM performs a constraint negotiation. Constraint negotiation may involve relaxing constraints by a particular KM. If constraint negotiation fails then system goes into a deadlock and alerts the KMs. Constraint negotiation can be performed at several levels. In the current system it will be assumed that refinement of levels in the Solution Blackboard partition occurs only after appropriate interaction (interface) constraints are satisfied.
7. If above process succeeds then Strategic KM sends a message to the Solution Blackboard partition, at which stage the details required for the next level in the Solution Blackboard partition are set up and appropriate KMs are activated.

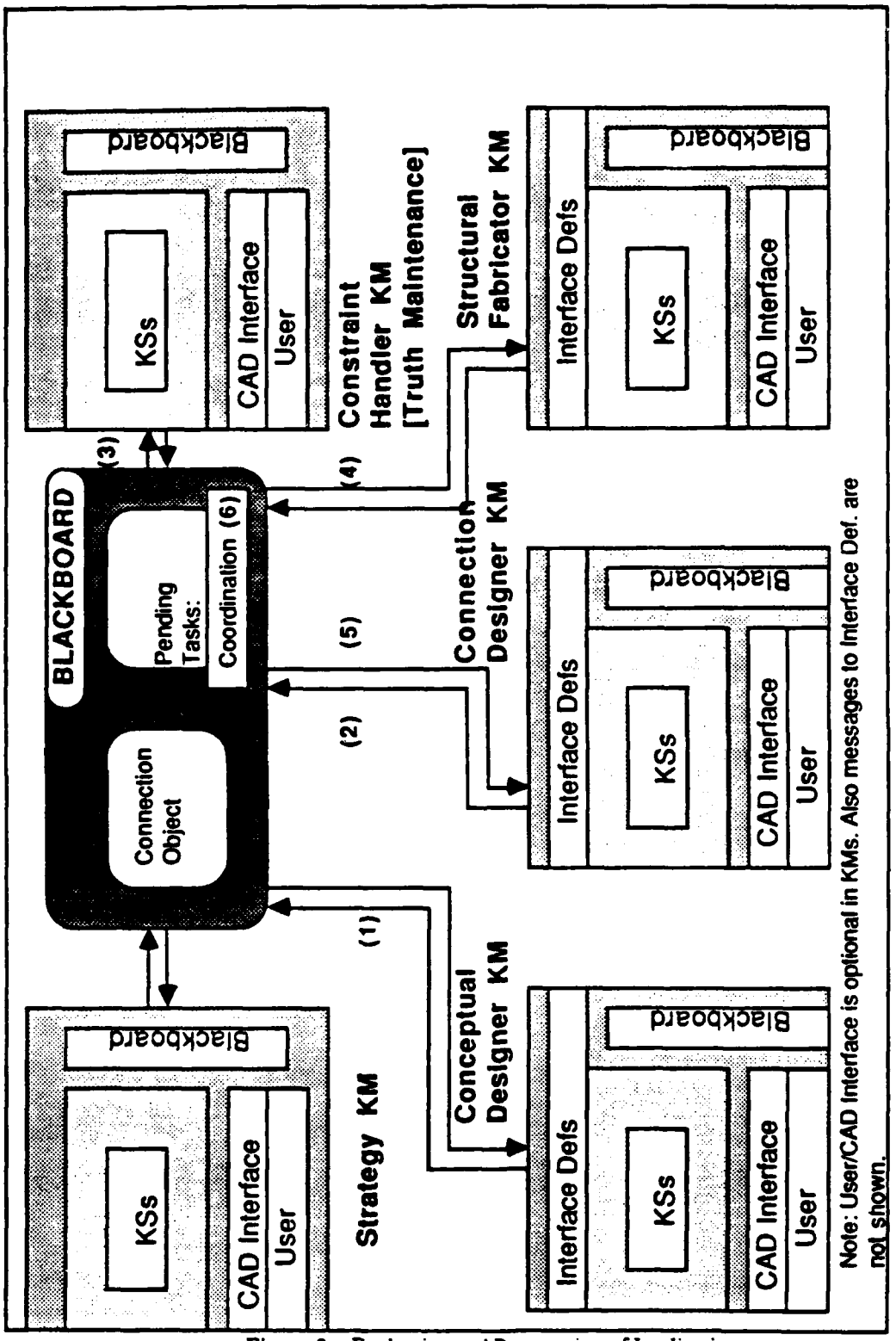


Figure 3: Evaluation and Propagation of Implications

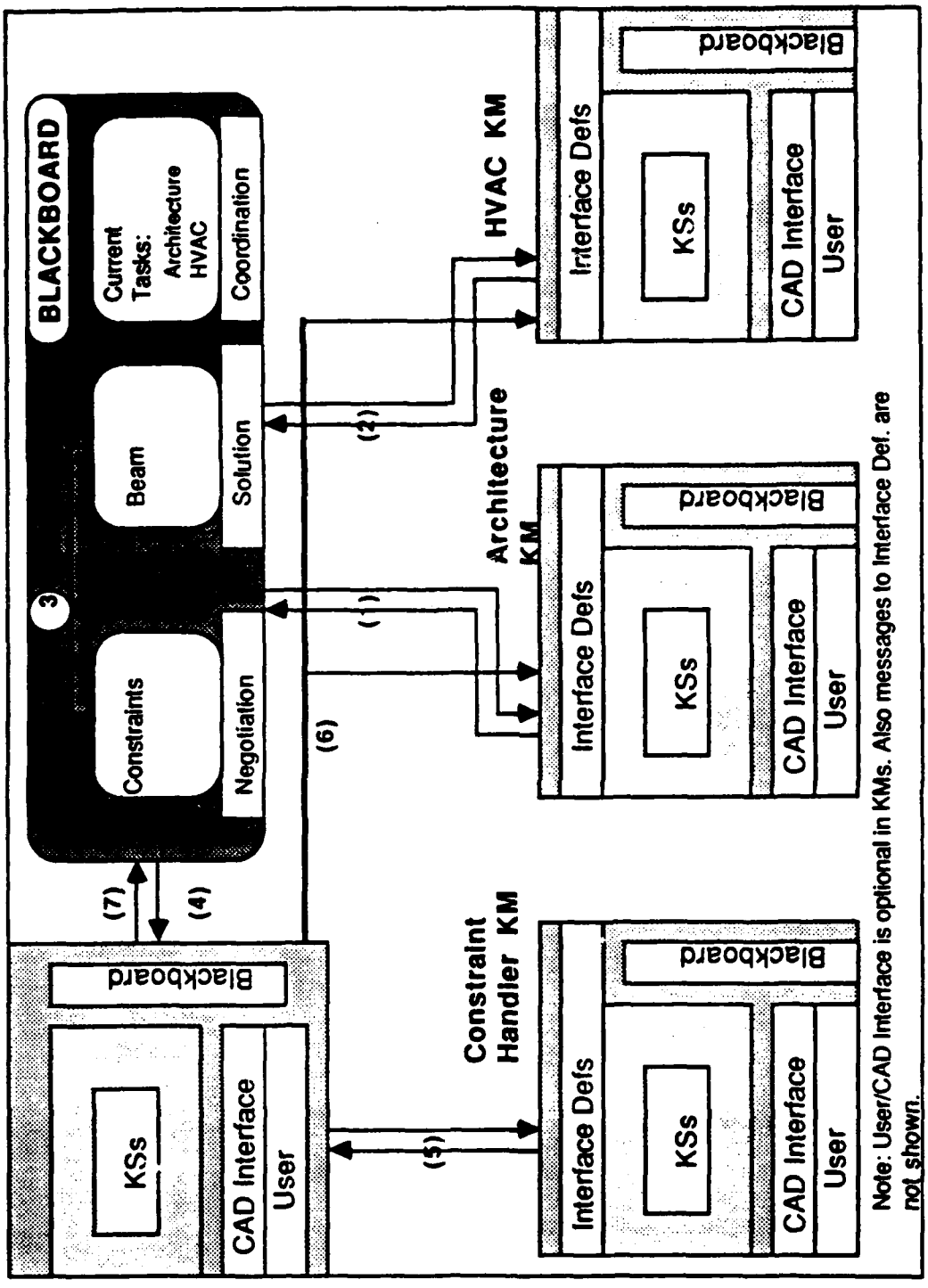


Figure 4: The Constraint Negotiation Process

4.3 Blackboard: Global Database

The purpose of the Blackboard is to: 1) provide a means for storing information that is common to more than one KM; 2) facilitate communication and coordination; and 3) ensure that designs and plans generated during design and construction are consistent.

The Blackboard in DICE is partitioned into: Coordination (CBB), Solution (SBB) and Negotiation BB (NBB) (Figure 5).

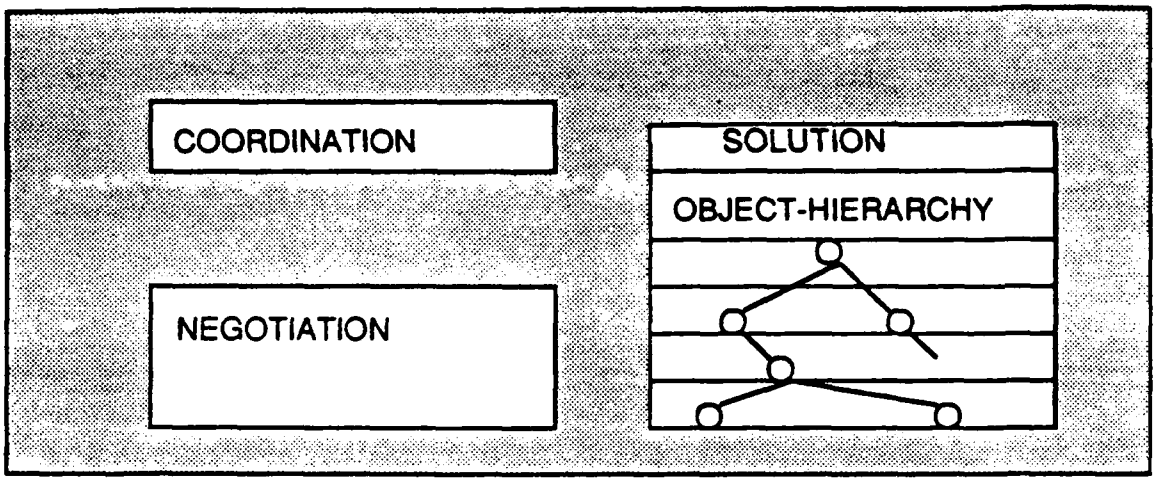


Figure 5: The Blackboard

4.3.1 Coordination Blackboard Partition

The Coordination Blackboard partition (CBB) contains the bookkeeping information needed for the coordination of KMs.

4.3.2 Solution Blackboard Partition

The Solution BB partition (SBB) is divided into levels (object-hierarchy). Each level contains objects that represent certain aspects of the engineering process (design and construction). The SBB does not contain all the information generated by all KMs, only information that is 1) required by more than one KM, and 2) useful in the engineering process is posted on the SBB. For example, the 3D space level will contain objects that represent spaces allocated to structural systems, piping systems, mechanical systems, etc. This level can be reduced to detailed levels, such as system and component levels.

The objects in SBB can be connected through relational links, where the relational links provide means for objects to inherit information; these relationships provide a framework to view the object from different perspectives. For the purpose of this work, the following relationships will be used in the SBB: *generalization (IS-A)* for grouping objects into classes, *classification (INSTANCE)* for defining individual elements of a class, *aggregation (PART-OF, COMPONENT)* for combining components, *alternation (IS-ALT)* for selecting between alternative concepts, and *versionization (VERSION-OF)* for representing various versions of an object. The semantics of these relationships are provided in [35]. Various planes that depict these relationships are shown in Figure 6.

The objects also contain justifications, assumptions, time of creation, creator, constraints, ownership KM, other

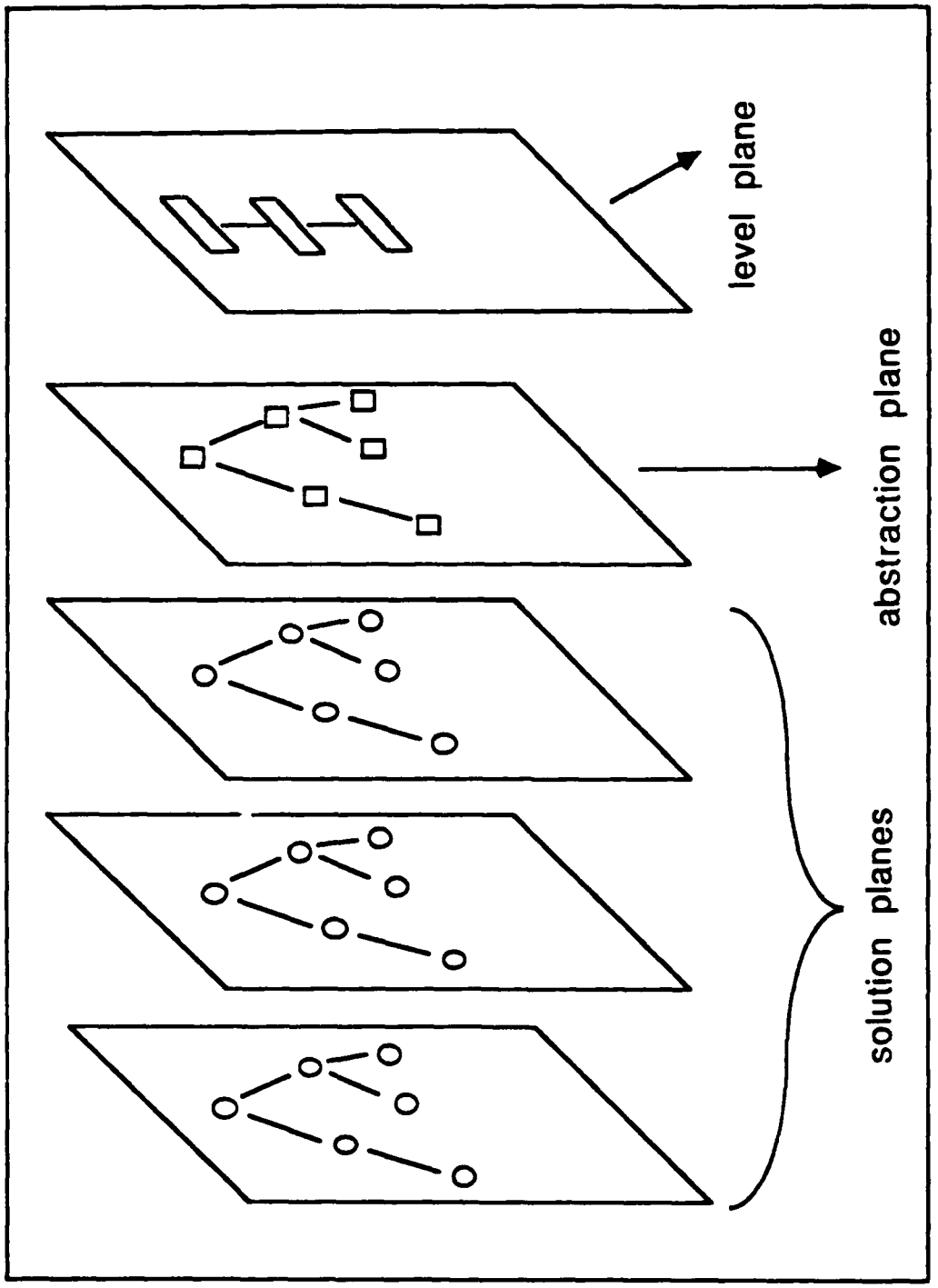


Figure 6: *Different Planes in the SBB*

concerned KMs, etc. The justification information will provide a designer's rationale and intent for the creation of the object. Assumptions made during design and construction are also stored with the object. For example, the architect, while placing the structural elements, may assume certain spatial characteristics for the HVAC systems. He may record this assumption and the rationale for such an assumption in the objects denoting the appropriate structural elements and the HVAC system. In DICE, *status* facets are associated with data attributes (slots). The *status* facet, for example, can take the following values: *unknown, assumed* and *calculated*. Additional slots will be needed for the source of data and its change, uses of data, assumptions made, etc..

Associated with these objects are methods which provide a means for: 1) performing some procedural calculations; 2) propagating implications of performing some actions, for example if the status (assumed or actual) or the value for a particular object changes then these changes can be broadcast to all concerned KMs; 3) helping to perform the coordination process. For example methods can be used as demons to perform the following construction related tasks:

1. **Estimating**, which involves continuous cost forecasting capabilities, from early estimates to detailed costs considering the equipment that will be available. This estimating will start with material and quantity modeling based on building standards for tenant work, and would first be updated with characteristics of the tenant. As layout work proceeds, material and quantity estimates would be updated.
2. **Scheduling**, which is similar in structure to Estimating, and uses much of the quantity data developed from the estimate forecast, passed to it with messages.
3. **Constructability**, where constant critics look for incompatible materials, space use, construction space needs, equipment requirements, etc.

Knowledge for all of these inputs will come from working with expert on all phases of the project, owner, designer and constructor. Further details of the use of methods in the communication process are provided in Section 4.2.

A typical object that resides in the SBB is structured as follows:

```
SBB-Object
        NAME:
        VALUE:
                status:
        CREATED-BY:
        JUSTIFICATION:
        PART-OF:
        IS-ALT:
        VERSION-OF:
        VERSION-NO:
        OWNED-BY:
        CONCERNED-KMS:
        CONSTRAINTS:
                range: (IS-A CONSTRAINT-OBJECT)
        -------- (and so on)
        METHODS:
```

4.3.3 Neogotiation Blackboard Partition

The Negotiation BB (NBB) could be viewed as consisting of two parts. The first part contains various interaction constraints that are imposed on the designed object. These constraints are developed during the definition of various

levels in SBB[7]. The second part consists of a trace of the negotiation process.

An object describing an interaction constraints is:

```
Constraint-Object
        CONSTRAINS:
                range: (IS-A SBB-OBJECT)
        INTERACTION:
                status:
                range: (IS-A INTERACTION-CONSTRAINT)
        OTHERS: (as needed)
        METHODS:
```

The *status* facet can take values like *satisfied*, *suspended*, *violated*, etc. A taxonomy of these constraints can be defined by the user. Adequate facilities will be provided for the user to incorporate these constraints.

4.4 Knowledge Modules

The Knowledge-base (KB) consists of a number of Knowledge Modules (KMs). Each of these KMs are further decomposed into small units called Knowledge Sources (KSs). The architecture of most KMs is similar to the overall architecture of DICE, i.e., knowledge is distributed among several objects (or KSs) and communicate through message passing. KSs can also be decomposed into smaller units, if desired. Thus the KB reflects the *hierarchical design process*.

Some KMs may incorporate both *textbook* and *heuristic* (surface) knowledge, while other KMs may include *fairly deep* knowledge. Surface knowledge consists mainly of *production rules* encoding empirical associations based on experience. This type of knowledge is useful for setting interface constraints between disciplines and between levels of design interaction. In a system with deep knowledge, both causal knowledge and analytical models would be incorporated. A fully deep system may be difficult to realize with the current state of the art of KBES. However, it is possible to encode analytical models. In this study, the term fairly deep knowledge will be used to denote analytical models.

The KMs, although distributed, can be classified into the following categories: the *Strategy*, *Specialist*, and *Quantitative KMs*. These KMs are briefly described below.

- **Strategy KMs** analyze the current solution state to determine the course of next action. A scenario using the Strategic KM is described in Section 6.2. Since this level may used to control various tasks, such as the activation of Specialist KMs during the coordination process, it comprises the *task control knowledge*.
- **Specialist KMs** contribute to various stages of design and construction (or manufacturing). Most KMs at this level are KBES that have a local *Blackboard* which may be divided into various levels of abstraction, and several KSs that interact with the local BB. The possible KMs that could be used in DICE for the domain of interior finishes are:
    1. *Architectural Designer*, for layout and finishes, including flooring and ceiling systems, etc.
    2. *HVAC*, for heat load calculations, duct layout, diffusers, etc.

---

[7]Constraints in engineering design can be of two types: constraints local to the object (designed) and interaction (interface) constraints that several objects should satisfy simultaneously. An example of a local constraint is *Beam.bending-stress should be less than 0.66*Beam.material.yield-stress*, while the example of an interaction constraint is *Pipes greater than 2 inches cannot go through steel beams or columns.*

3. *Lighting*, for layout, lighting levels, heat generation, etc.

4. *Plumbing*, for layout, etc.

5. *Construction Planning*, for schedules, costs, constructability checking, etc.

6. *Structural*, only for detailing attachments.

Individual KMs will, most probably, be residing on different machines and will make extensive use of networking protocols for communicating with the Blackboard.

- **Critic KMs** contain the knowledge and mechanisms to check the consistency of the designs. Constraint Handling KM is an example of a Critic KM.
- **Quantitative KMs** contain the analytical knowledge and reference information required for analysis and design. These KMs are typically comprised of algorithmic programs and databases. Quantitative KMs comprise the algorithmic knowledge of the domain. The Specialist KMs mostly communicate with the Quantitative KMs through a Blackboard that is local to the Specialist KM.

The user forms an integral part of these KMs. An important issue in the development of KMs is the man-machine interface and how the information generated by the user is transmitted to other KSs. We assume that the user interacts with the computer through a high resolution bit mapped display (or appropriate system). Hence, there is a need to provide the appropriate semantic translations from the information provided by user to the form required by other KMs or KSs. In DICE this is achieved by the *interface definition* module. Further changes made by the user will be recorded in the local and global Blackboards (if needed) and appropriate actions triggered. Hence, the user can be viewed as a KS taking part in the solution process.

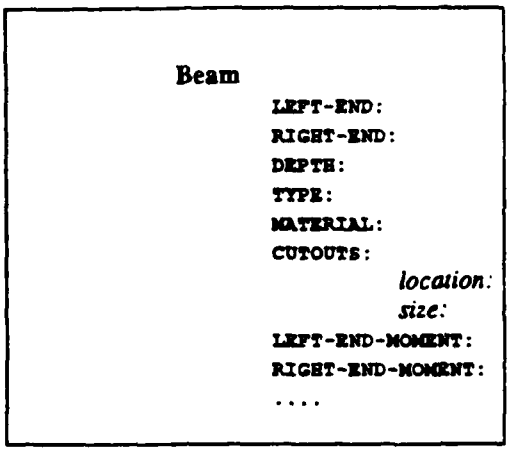
The KMs (mostly Specialist and Strategy) can post and retrieve information from the global Blackboard. However, an object (and associated attributes) in the Blackboard can have varied connotations (most semantic) in different KMs. Hence, there is a need to define the semantic mappings (translations) between the objects in the KMs to the objects in the Blackboard. As an example, consider the object **Beam**. In the architectural KM, the beam may be defined as follows [17]:

```
Beam
        LEFT-END-COLUMN-LINE:
        LENGTH:
        WIDTH:
        MATERIAL:
        TYPE:
        DEPTH:
        VIEW
        ....
```

While the same object may be defined in the HVAC KM as:

```
Beam
        L-END:
        R-END:
        D:
        TYPE:
        MATERIAL:
        R-END-MOMENT:
        ....
        METHODS: possible-cut-outs
```

In the Blackboard, the same object may be defined as:



In DICE the methodology used in [17] is being adapted for developing the necessary semantic translations.

4.5 The Negotiation Activity

The negotiation process takes place once a conflict is detected by the Truth Maintenance System. Negotiation is achieved through the help of the Strategic KM.

Conflicts can occur either due to interface constraint violation or due to contradictory modifications of a single object. For example, a HVAC KM can decide to place pipes at the same location that the architect decided to place a beam. These conflicts can only be detected once the two designs have been posted and sufficient constraint propagation and/or modeling has been performed. Another type of constraint violation occurs when a KM changes to a partial solution posted by another KM. The two participants may or may not have similar roles in the system. For example, two architects may disagree on the location of the walkway, or the HVAC KM might want to change the depth of a beam posted by the structural engineer in order to put some pipes through it.

Once a conflict has been identified, a two fold mechanism helps the conflicting KMs in negotiating towards a mutually agreed solution. Constraint relaxation is first attempted and is followed by goal negotiation in case of failure.

The first attempt to negotiate involves traditional constraint relaxation techniques and implements compromise bargaining. Assuming that the conflict is due to constraint violations of certain design parameters, the system can act as a third party and offer compromise values to each party until an agreement is reached. In order to allow this scheme to function properly, each value posted on the Blackboard has to be accompanied by a constraint. Each constraint must specify the range of possible values. These constraints can be either *soft* or *hard* constraints.

The soft constraints can be negotiated upon and are not imperative. For example, an architect can decide that the aesthetic proportions of a particular beam imposes a certain width to depth ratio, while the structural engineer finds another ratio guided by strength requirements.

The hard constraints always have to be satisfied. These are usually bounds imposed by code regulations or physically existing constraints such as a neighboring building, etc. In this case, no relaxation is possible and another negotiation mechanism has to be triggered. In the case of hard constraints, the range of possible values is automatically set to a unique value. If the system detects a conflict involving soft constraints which have a non-empty range intersection, it can directly modify these values and notify the KM of the change. If the ranges do not

show compatibility, the Strategic KM has to contact the conflicting KMs and initiate constraint relaxation. In some cases, it is hoped that the ranges themselves have soft constraints and that a compromise can be found.

The second set of techniques involves the redefinition of design goals. The KMs are asked to negotiate on a more abstract plane. It is considered that the set of conflicting constraints are the concrete expression of an abstract hierarchy of goals. At the root of this hierarchy is the goal of designing and constructing the artifact for which the design team has been set up. Each participant develops his own hierarchy of personal goals (see Figure 7 and Figure 8). By helping the KMs find an agreement goal and developing a common set of more detailed goals, the system achieves integrative agreement.

The example of the Hyatt Regency walkway connection design will serve as an illustration of the multiple levels at which this goal negotiation can take place.

At the lowest level, the conflict is in the goals assigned by each participant to the connection and the rod(s). The designer wants the connection to transfer the load of only one walkway, and wants the rod to transfer the load of the two walkways to the roof. The fabricator wants each rod to transfer the load of only one walkway, and wants the connection to bear twice the load. Goal negotiation at this level can lead to a solution where two rods are connected at some level below the box-beam (see Figure 9). Then, each rod transfers the load of only one walkway and so does the connection.

At the next level, the system suggests that the connection need not be designed at all. That means that a different solution can be sought, avoiding the connection of the two parts of the rod and the box-beam. Such a solution could consist of hanging each walkway from its own rod directly to the roof as in Figure 10. The aesthetic of this solution might not please the architect, but this is another conflict story!

The next level of abstract negotiation would consist of avoiding the design of walkways. This could be done by providing fast convenient elevator service or preventing direct level to level access across the lobby, etc. Even further, the system could suggest to get rid of the lobby and design the whole building differently. By rearranging the layout, a new solution might be found which doesn't require a lobby. Finally, the participants might come to the conclusion that they have no common goal and that they do not wish to design and construct the building. Some negotiations are bound to end that way.

5 Current Status

During the initial stages our major focus has been the development of: 1) utilities for defining the SBB object hierarchy, 2) transactions for posting, modifying and deleting information from the Blackboard, 3) a simulation program to demonstrate the utility of DICE, and 4) a prototype which involves the automatic generation of construction schedules from an architectural drawing. DICE is being implemented in a hybrid programming environment called PARMENIDES/FRULEKIT; PARMENIDES/FRULEKIT supports programming in frames and rules and was developed in LISP at Carnegie-Mellon University by Carbonell and Shell.

These topics are briefly described below.

5.1 Graphic Definition of Objects

The user can interactively define class objects[8] in the Blackboard and the KMs. Classes can either be created in

[8] A class denotes the grouping of objects (instance or class) which have similar characteristics, while an instance is a particular individual which belongs to a class.

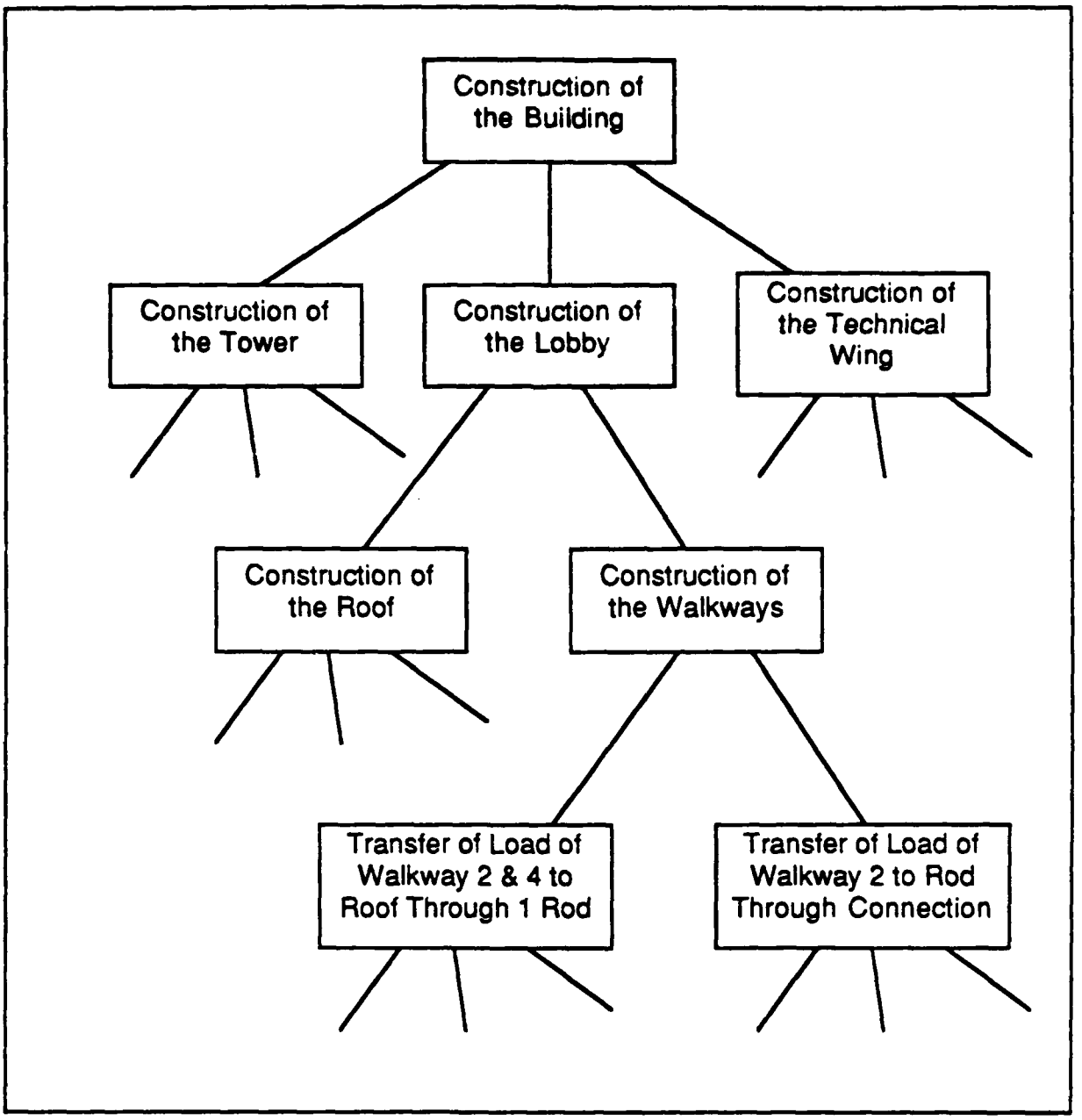


Figure 7: The Designer's Goal Hierarchy

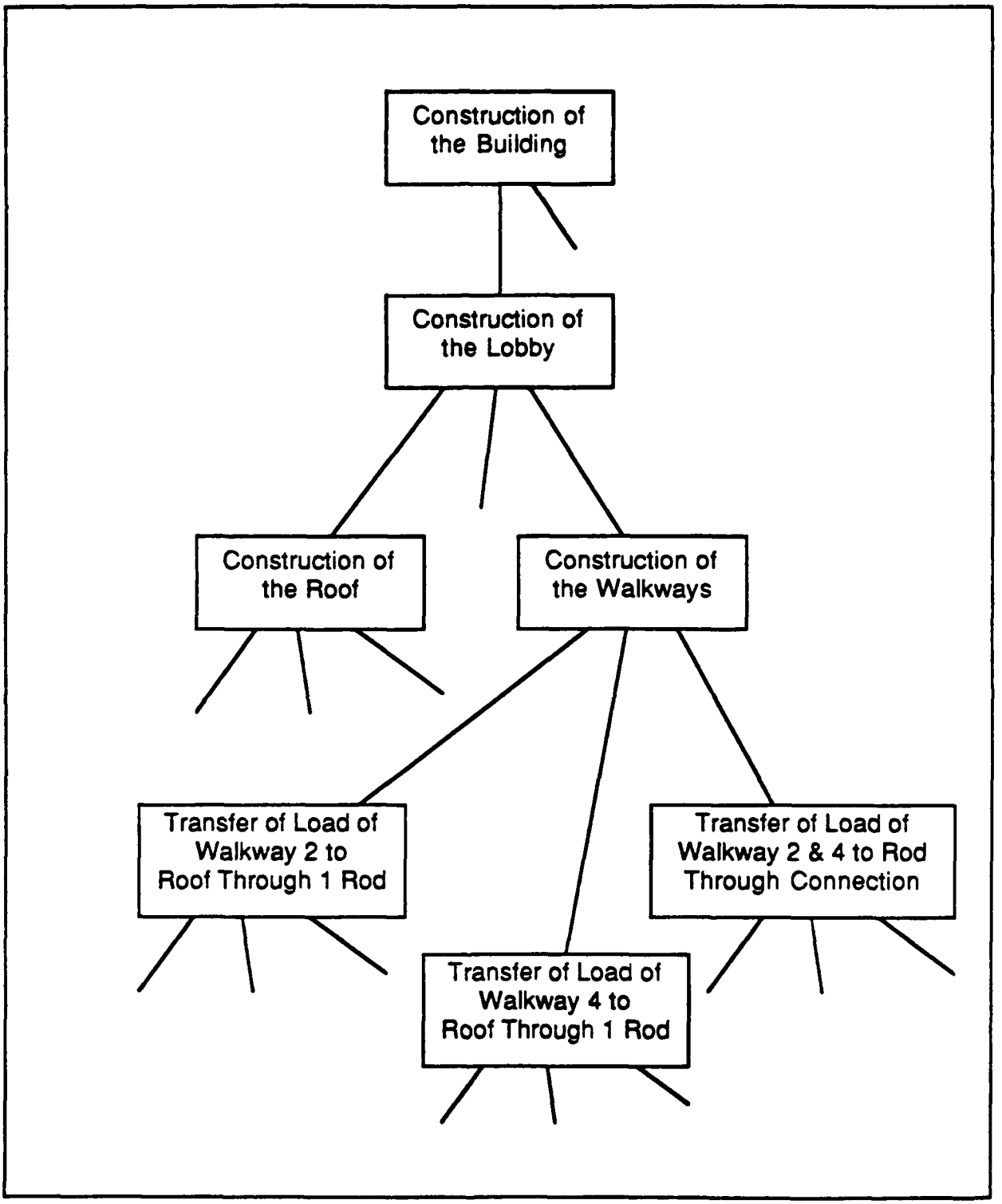


Figure 8: The Fabricator's Goal Hierarchy

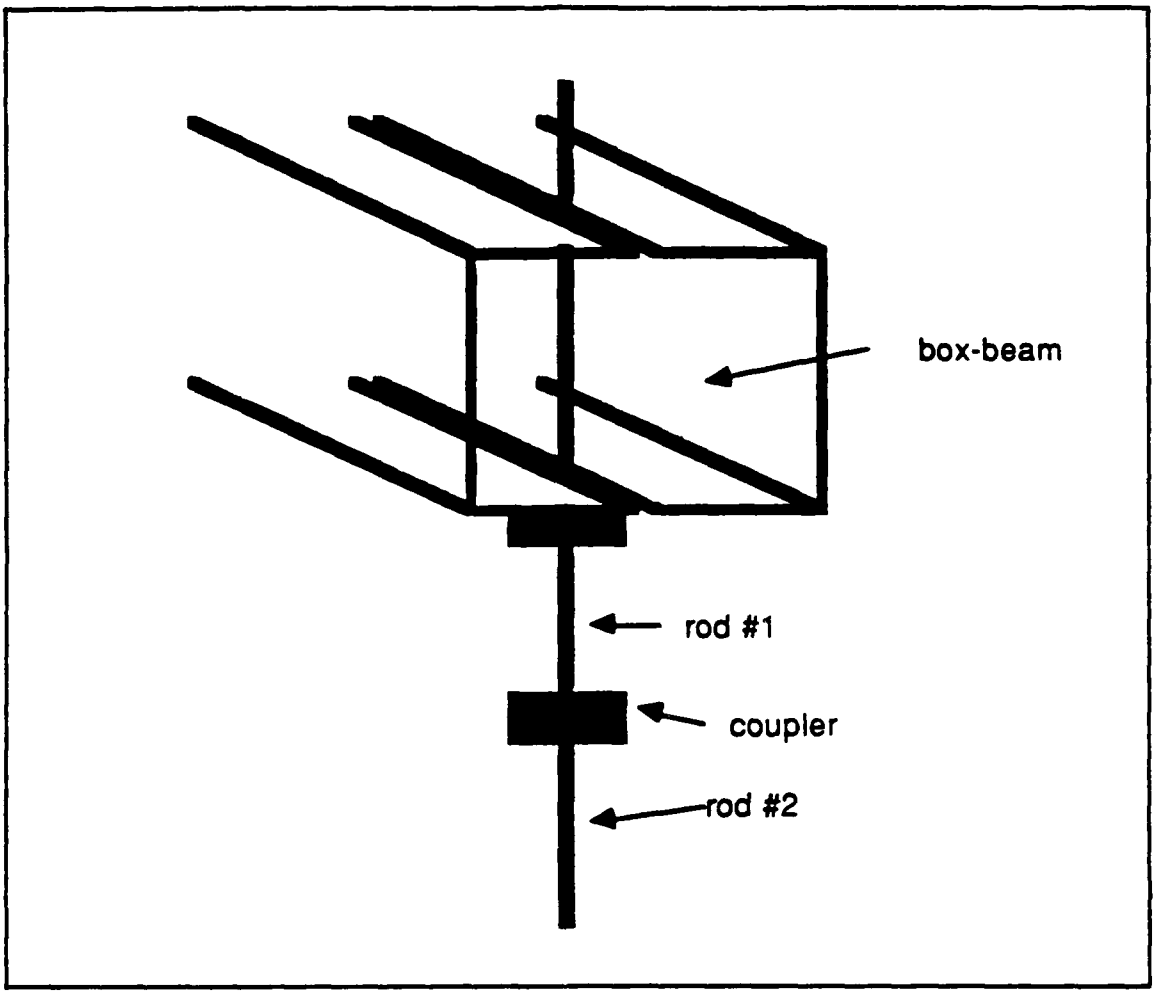


Figure 9: The Connected Rods Solution

LISP or through a menu interface provided to the user. Each class has a name, several slots which describe various attributes, and associated with each slot are facets which provide further information about the slot; the facets also contain methods.

A class object is created by clicking on CREATE-CLASS in a menu on the screen. After creating a class object, the user can display the class using the DISPLAY-CLASS option, as shown in Figure 11. In Figure 12, the class **Build1** is made a subclass of the **Hierarchy-object** class, which becomes **Build1**'s superclass. When this link is made, all slots in the **Hierarchy-object** class are inherited by **Build1**. In addition, the user can create new slots or delete slots. For example, in Figure 13 the user created two slots, namely NAME and HAS-PARTS. Slots can be faceted or non-faceted. The creation of facets is shown in Figure 14.

Instances of a class can either be defined interactively through a menu or by LISP functions. For example the function *(create-instance 'Build1 'Hyatt-regency)* would create an instance, **Hyatt-regency**, of **Build1**.

In addition to defining classes and instances, the user can also display the class hierarchy, in the form of a tree. Nodes in the tree depict classes and instances. Each node is displayed in a box with the name of the class or instance. If the name does not fit in the box then it is abbreviated. The user can drag the mouse pointer over the tree

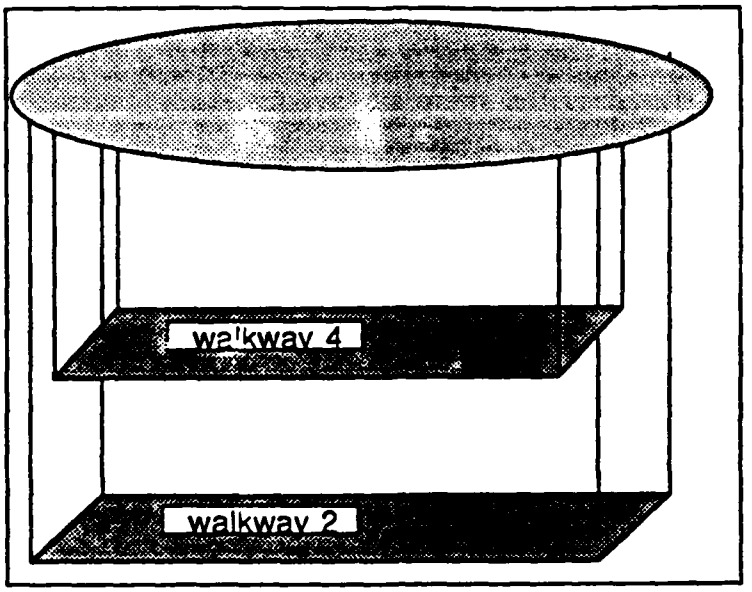


Figure 10: The Split Connections Solution

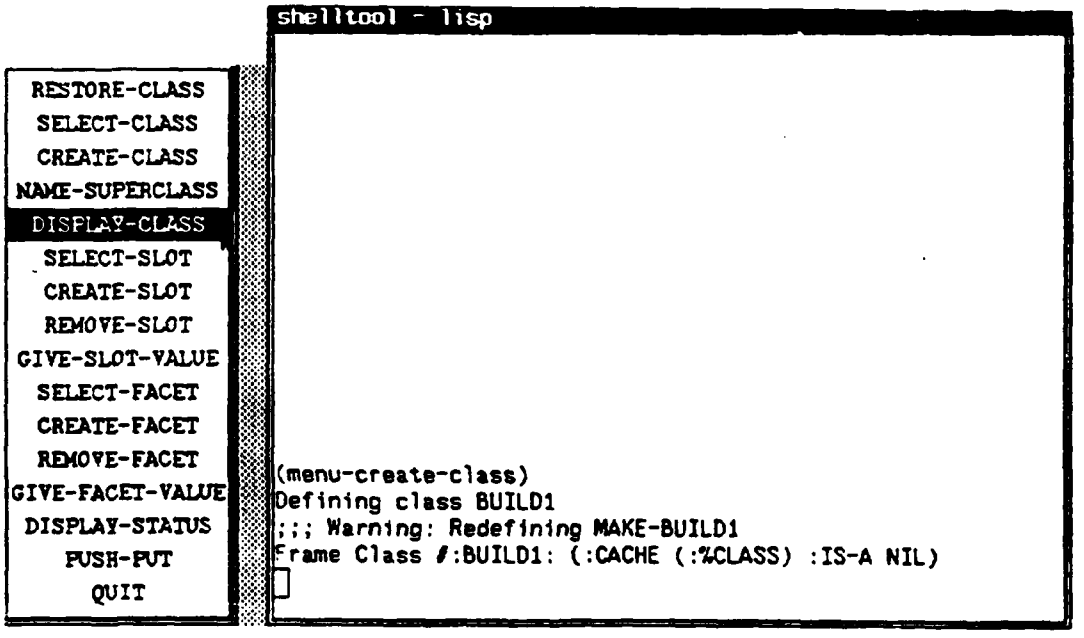


Figure 11: Displaying a Class

to an appropriate node. This will display the full name of the node (Figure 15 a). If the user wants more detail about

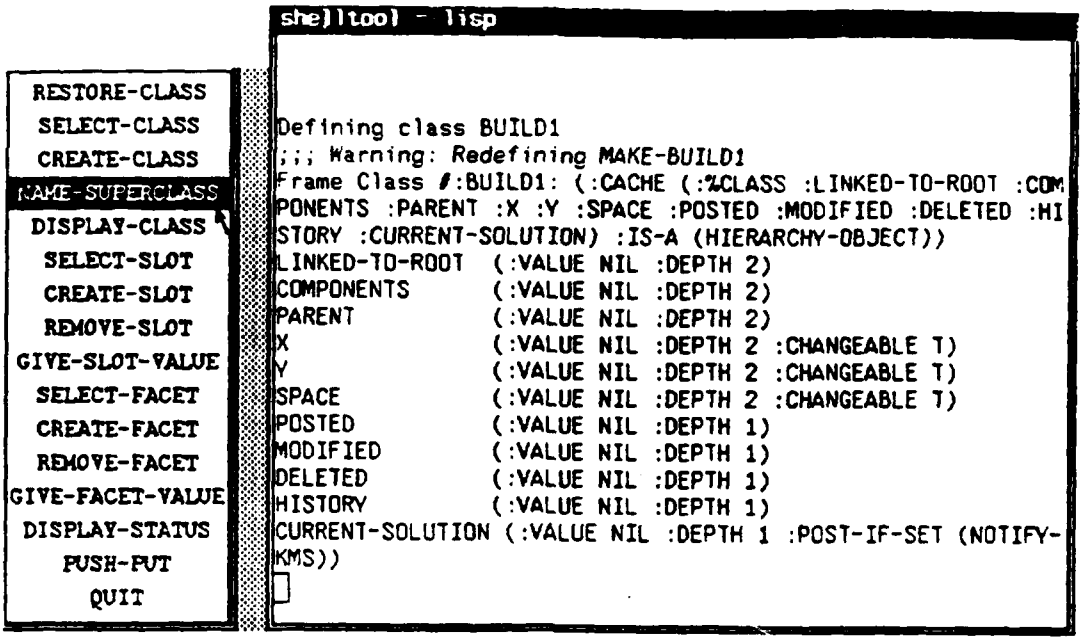


Figure 12: Linking a Class to its Superclass

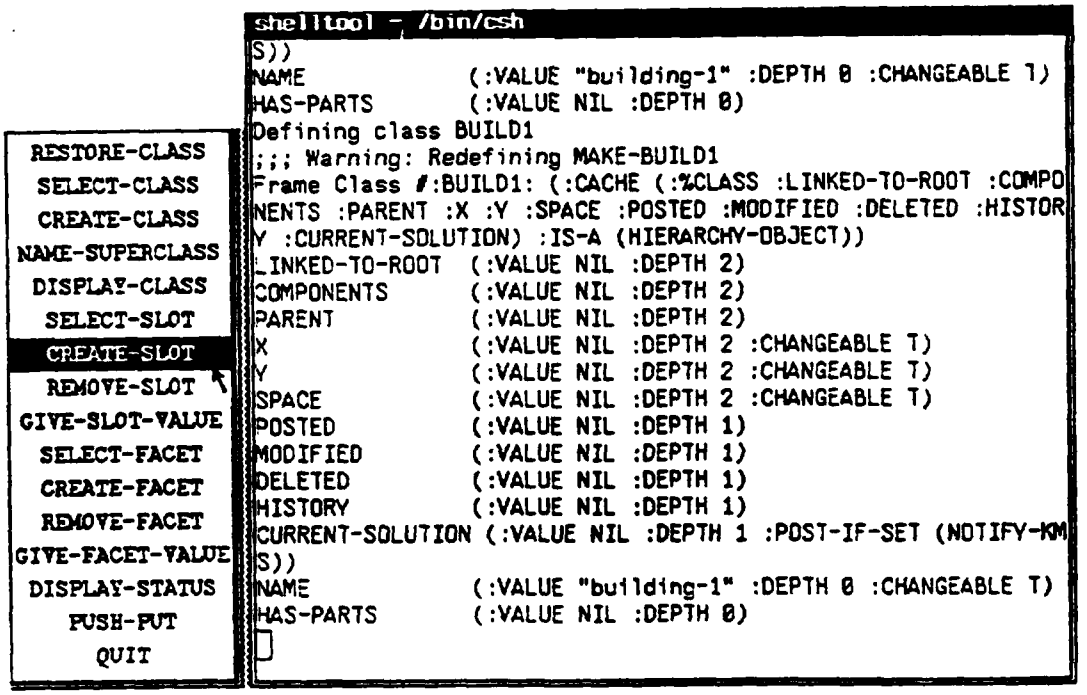


Figure 13: Creation of Slots

the node, he can click on the node and he will be shown the slots of the object corresponding to the node (Figure 15

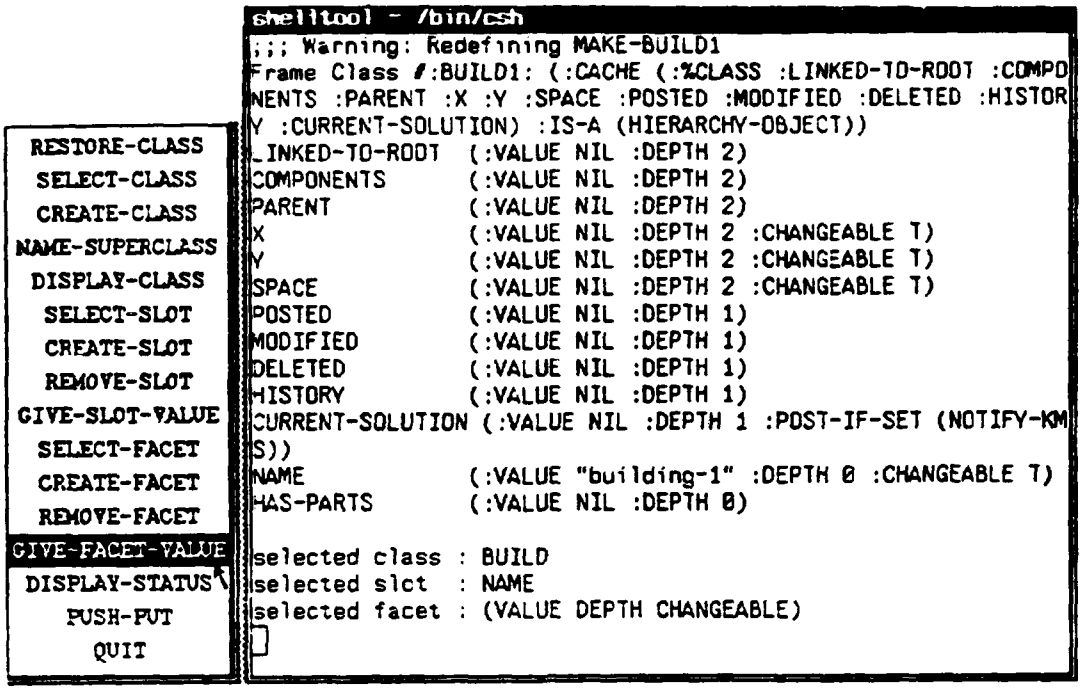


Figure 14: Creation of Facets

b). Facet information for any slot can be obtained by clicking on the slot (Figure 15 c).

5.2 Blackboard Transactions

Communication between KMs is achieved through the Blackboard. The communication channels are established in special slots of the object hierarchy in the SBB. Whenever a new KM is attached to DICE, its address is placed in a special frame in the Coordination Blackboard partition.

Currently, three types of messages can be sent to the Blackboard from the KMs (and in some cases vice versa). All messages are put in a mail-box object and processed sequentially. These messages are described below.

1. **Post** allows a KM to store an object or objects at the appropriate levels in the SBB. The syntax of post is: *(Post local-object remote-object &file)*, where *local-object* is the object or pointer to a tree of objects in a KM, *remote-object* is the object/level in SBB, *file* is the name of the file that *local-object* is stored in; the & sign indicates that the file name is optional and the system creates its own name if the file name is not provided. As soon as the Blackboard receives the posted message it accesses the appropriate *file* in the KM and updates the SBB. This process is depicted in Figure 16.

2. **Retrieve** gets the information from the SBB to a KM. The syntax of this command is *(Retrieve remote-object &file)*. If object does not contain any information, i.e., it has a value *nil*, in the SBB then the Blackboard relays a message across the network to the appropriate KM that can provide the information; it is assumed that objects in SBB contain the names of the KMs that can update the objects. The retrieving process is depicted in Figure 17.

3. **Delete** provides a KM the ability to delete information on the Blackboard (SBB). The syntax of delete is *(Delete remote-object)*. Delete does not erase predefined classes, but only removes the instances. This function is currently being updated.

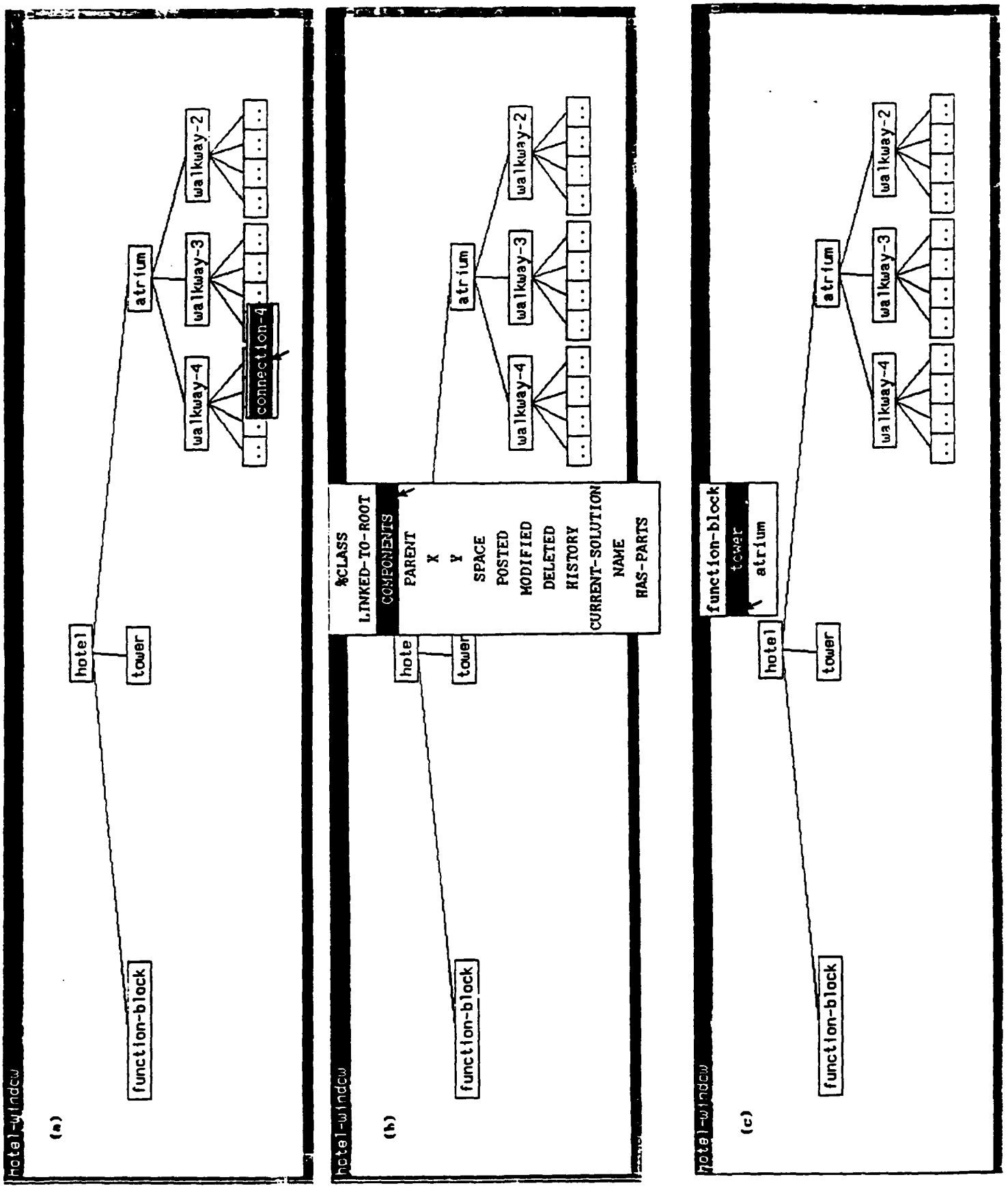


Figure 15: Display of the Object Hierarchy

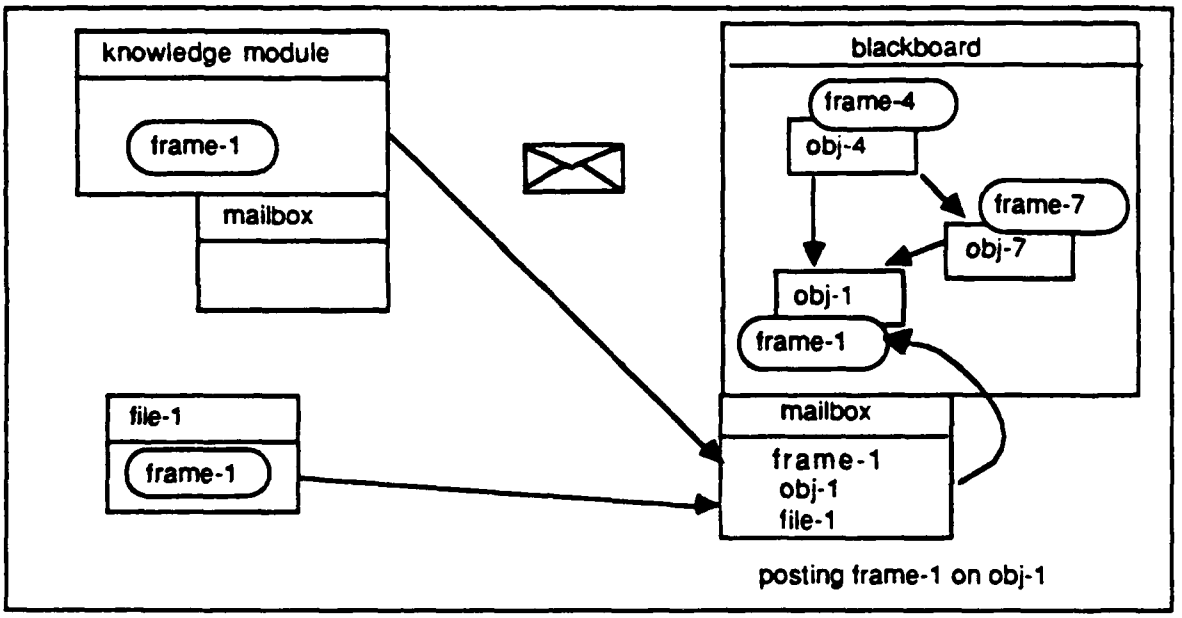


Figure 16: *Posting Information to the Blackboard*

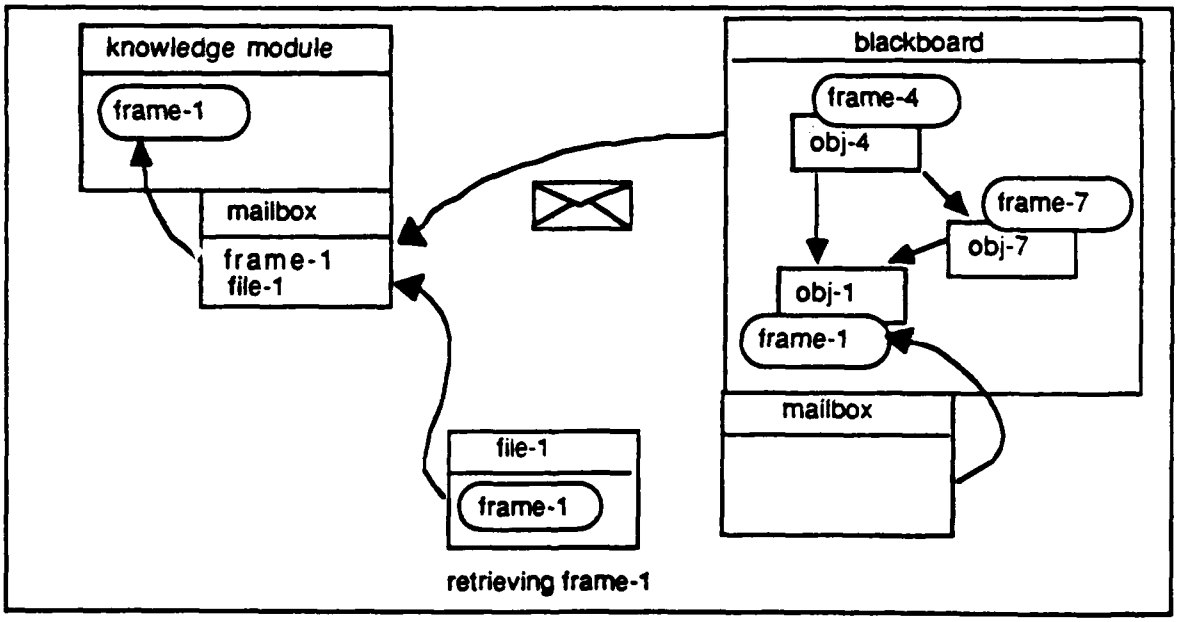


Figure 17: Retrieving Information From the Blackboard

5.3 A Simulation

A simulation of the Hyatt Regency design process was developed on two SUN computers to demonstrate some of the capabilities of DICE. A Blackboard, a Critic KM, a Constraint Manager KM, and a Strategic KM exist on the first machine (Figure 18), while a Connection Designer KM and a Structural Fabricator KM reside on the second machine (Figure 19).

The design-fabrication sequence is described below and shown in Figures 19 through 24.

1. Connection designer KM posts the connection design (denoted by 1-rod-connection) of the fourth floor walkway on the Blackboard (Figure 19 a).
2. Blackboard receives the design (Figure 20 a and b).
3. The connection object has a method that indicates that the connection design should be checked by the Critic KM. Hence, the Blackboard sends the connection design to the Critic KM (Figure 20 c and d).
4. The Critic KM replies that the connection design is acceptable[9] (Figure 20 e).
5. The Structural Fabricator KM is sent a message that a connection design has been completed and needs fabrication (Figure 21 a). The Fabricator retrieves the connection design, makes modifications and sends it back to the Blackboard (Figure 22 a, Figure 23 a and b).
6. Blackboard notifies the Strategic KM to check for possible conflicts (Figure 23 c).
7. Strategic KM retrieves the two connection (rod) designs (Figure 23 d) and sends it to the Constraint Manager KM to check for violation of interface constraints (Figure 23 e).
8. Constraint Manager KM notifies the Strategic KM that the designs are incompatible (Figure 23 f).
9. Strategic KM notifies this to both the Connection Designer and the Structural Fabricator (Figure 22 b and c, Figure 24 a and b).

The above simulation program was completed in December 1987. In the next section, a project that was undertaken during January 1988 to September 1988 to demonstrate the interface definition modules is described[10]. Implementation details can be found in [15].

---

[9] In the actual design the original connection design itself was faulty, but we assume here that it is an acceptable design.

[10] The NBB was not utilized in the above simulation.

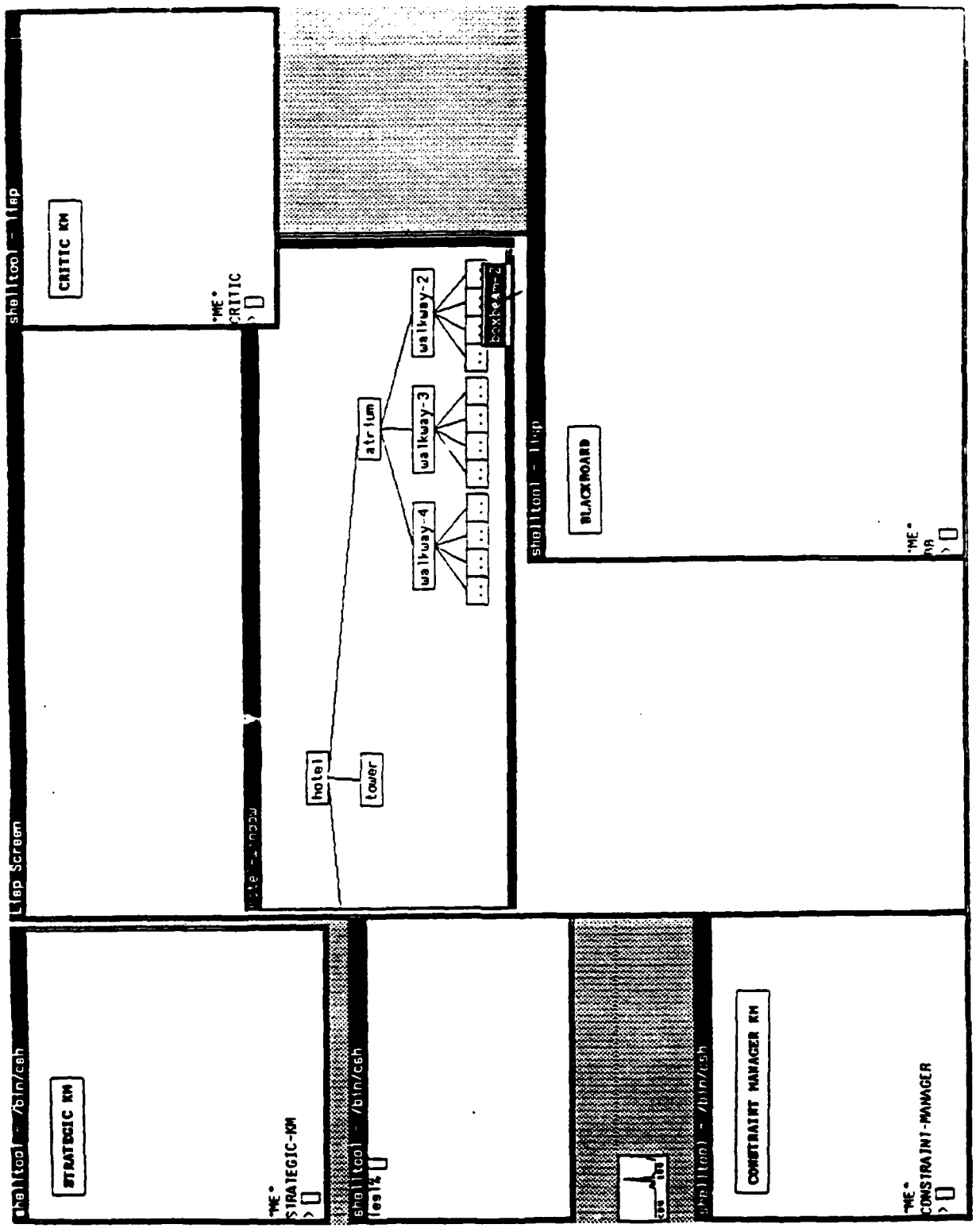


Figure 18: Set up for Simulation of the Hyatt Regency Design Problem

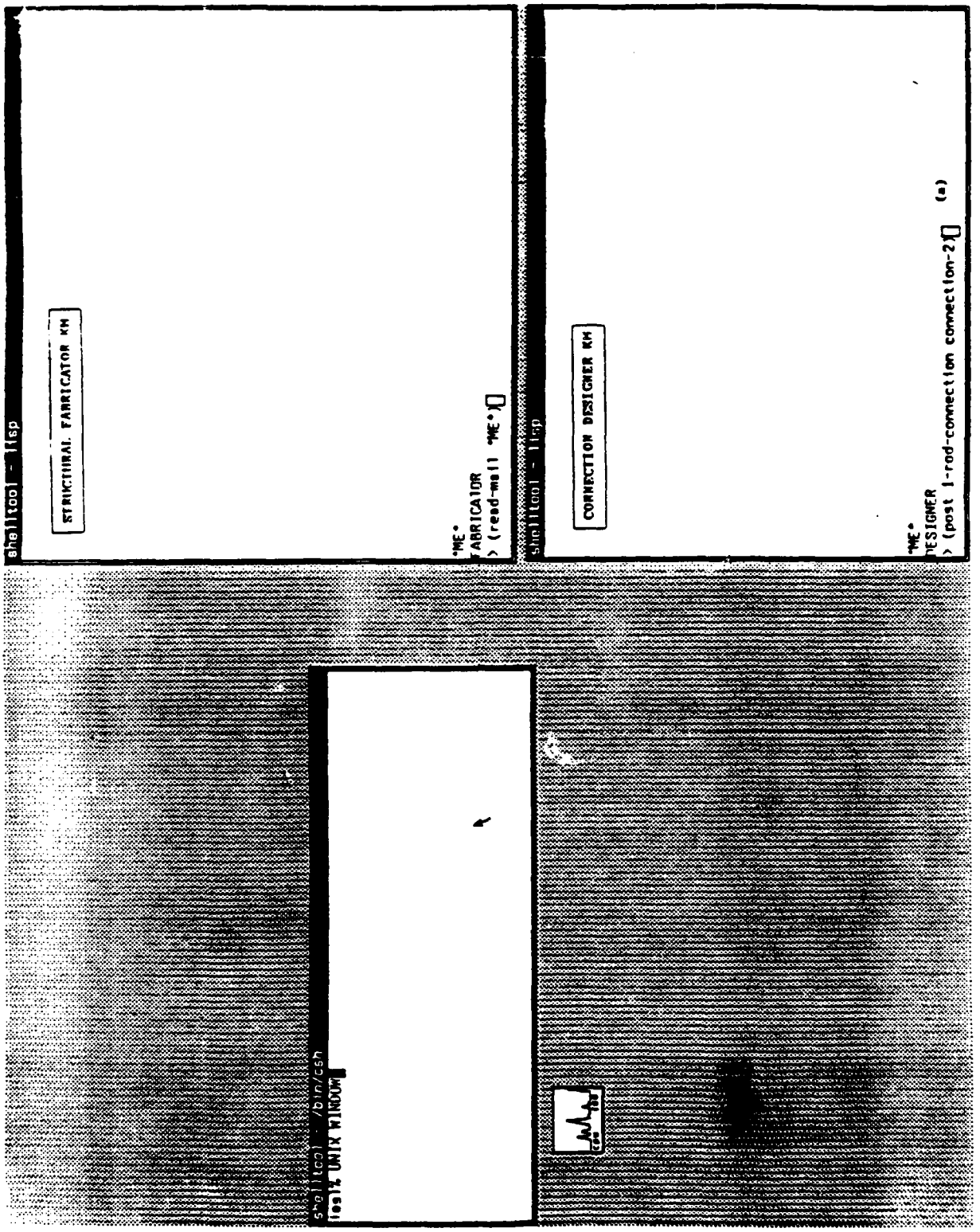


Figure 19: Set up for Simulation of the Hyatt Regency Design Problem (Ctd.)

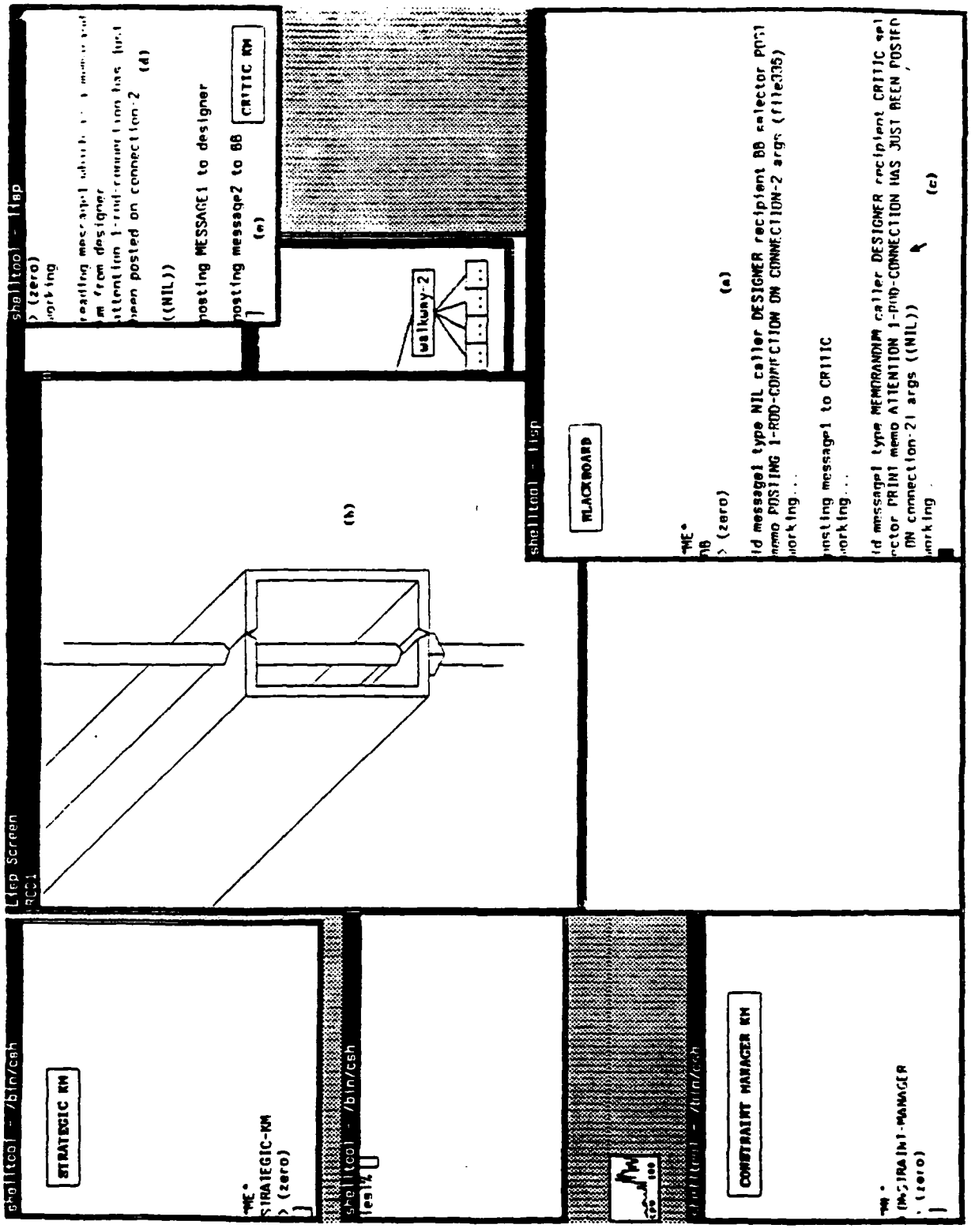


Figure 20: Simulation

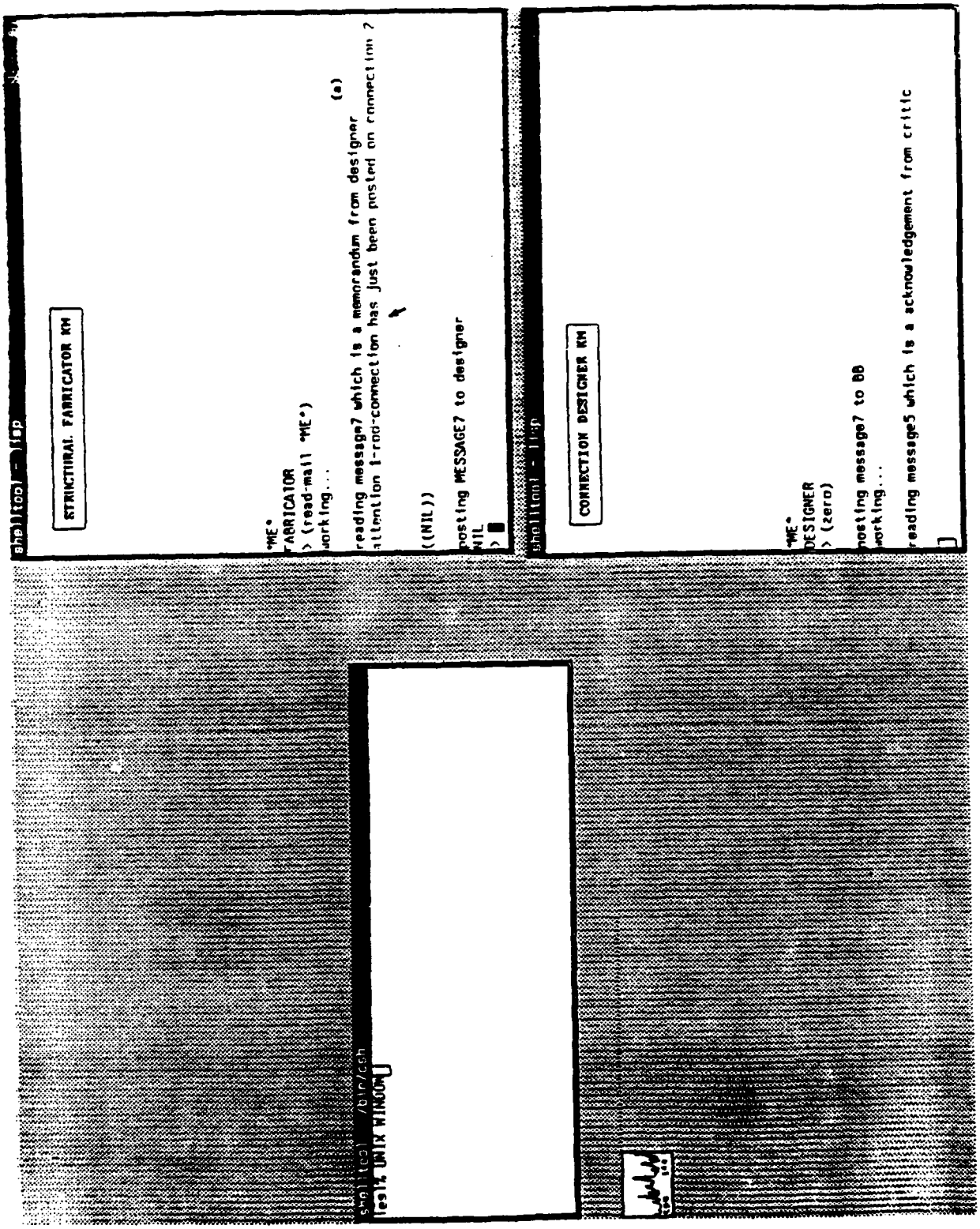


Figure 21: Simulation (Continued)

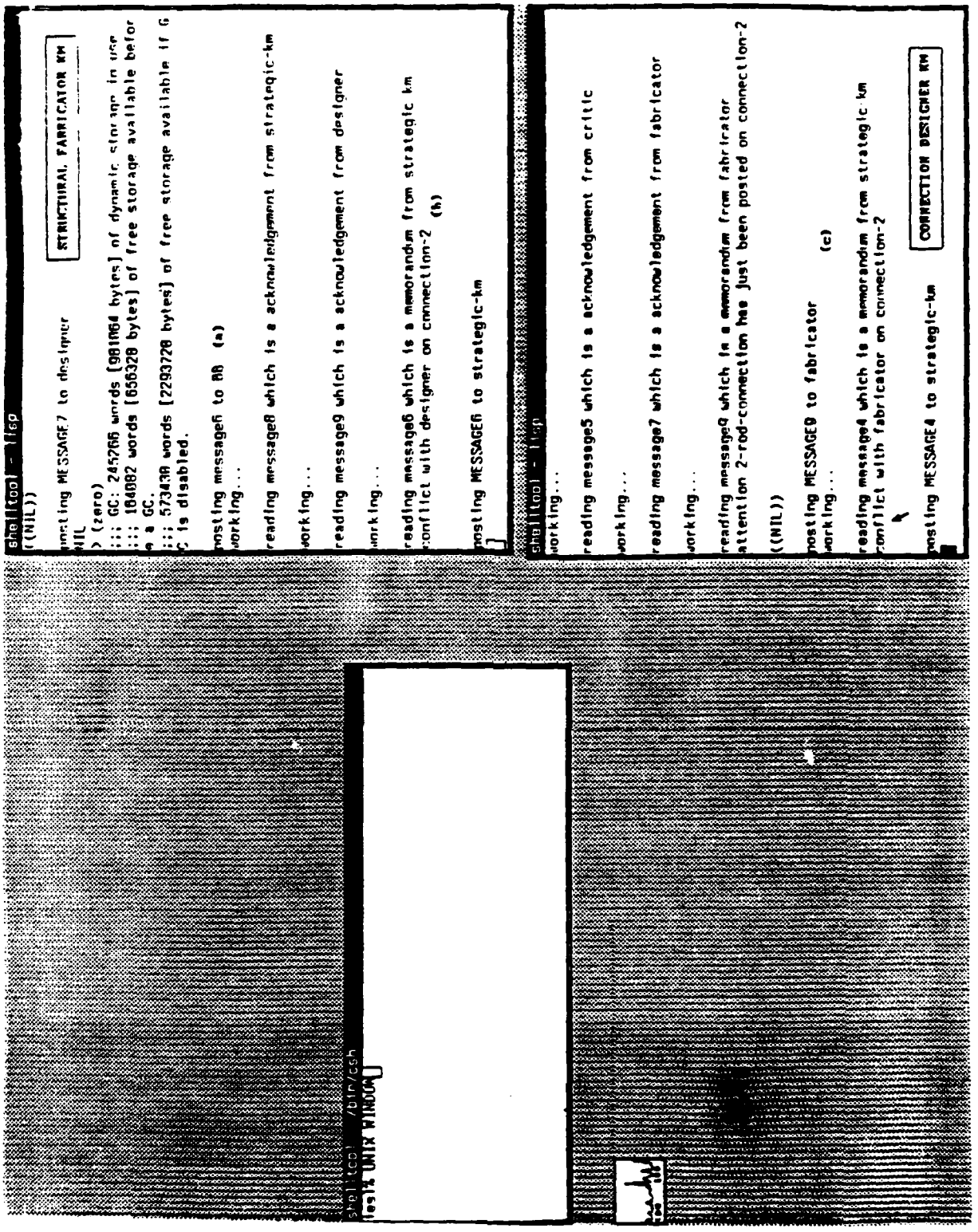


Figure 22: Simulation (Continued)

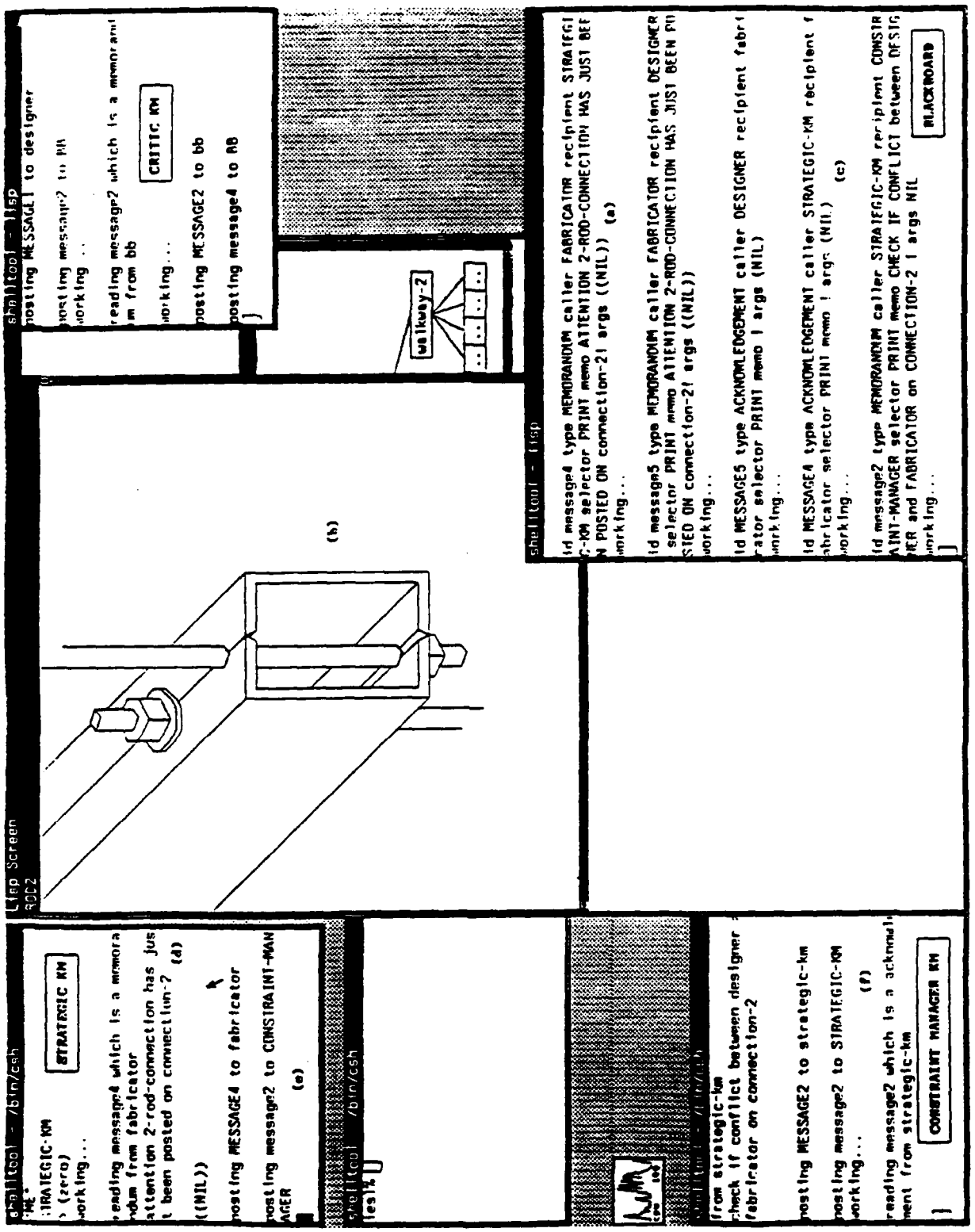


Figure 23: Simulation (Continued)

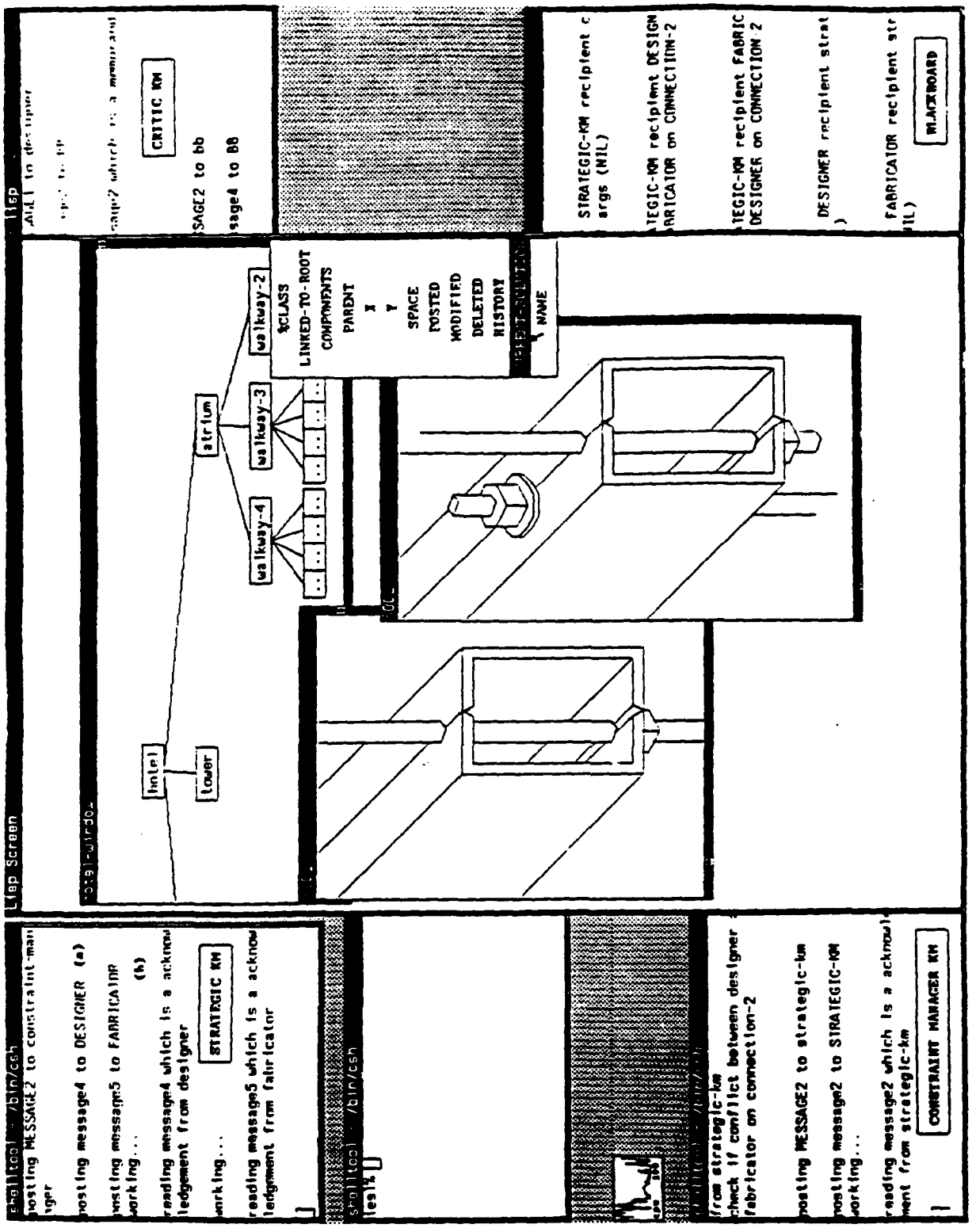


Figure 24: Simulation (Completed)

5.4 BUILDER in DICE

BUILDER automates the task of generating and maintaining schedules from architectural drawings. The initial version of BUILDER [5] was developed in KEE™, which is a hybrid knowledge-based programming environment [14, 19]. In this version, BUILDER had three major components - a drawing interface, a construction planning expert system, and a CPM algorithm - implemented as a layered knowledge-base, as shown in Figure 25. The various components of BUILDER are briefly described below.

1. **Drawing Interface.** The drawing interface layer provides for graphic input of an architectural plan. It is a menu-driven drafting system that incorporates the following features.
   a. Provides a convenient drawing system.
   b. Does the initial processing necessary to identify and classify the building components in a drawing, producing a representation of the drawing using a frame-based representation.
   c. Extracts the geometric features and produces a semantic network representation of the drawing; this semantic network representation links together the frame representation of building components.

   The friendly interface is facilitated by access to the underlying knowledge structures about building components. The menu driven system can automatically access the meanings of the symbols that it draws.
2. **Construction Planning KBES.** In an architectural drawing, the semantics of objects is normally not explicitly represented. For example there may be doors, walls, and plumbing in the drawing, but information about ordering materials for walls and doors, or having the plumbing inspected is not encoded. Neither is there any information about sequencing of tasks, or task durations, quantities, and costs. The first step in scheduling the job is to make a complete list of the tasks that need to be done. BUILDER utilizes an object-base, which is a database of engineering entities represented as frames (or objects), to complete the task list. Rules about construction methods are then activated to generate the precedence relationships between tasks. Next, BUILDER accesses a conventional database and generates an estimate of the quantities required and associated costs.
3. **CPM Algorithm.** An object oriented and a conventional CPM algorithms are implemented in BUILDER. The object oriented approach offers some efficiency and modularity over the traditional technique in project updating, reporting, and modifying. The standard CMP algorithm is implemented for initial scheduling efficiency.

In the second version of BUILDER - DICEY-BUILDER, we are implementing the above three components as three separate KMs, as shown in Figure 26. The purpose here is two fold: 1) to demonstrate communication between heterogenous KMS, and 2) to utilize this prototype to develop a protocol mechanism - similar to the Local Area Netework's OSI model - for the domain of building design and construction.

The Blackboard in DICEY-BUILDER is represented as frames in PARMENIDES, while the KMs are implemented in KEE. The translation to the Blackboard from a KM and vice versa is achieved by first transforming the frames to an intermediate representation language (IRL) and then translating from IRL to the appropriate KM; the syntactic and the semantic translations are similar to the approach described in [17]. The initial Blackboard structure is generated using the editing facilities described earlier (Figure 27). Figure 28 a shows an object in the DRAW-KM. The intermediate representation format, which is a list in the current implementation, is shown in Figure 28 b, while the corresponding Blackboard frame is shown in Figure 28 c.

6 Summary and Future Work

In this paper, we have described DICE, which is a collection of computer-based tools for cooperative engineering design. DICE facilitates coordination and communication in engineering design by utilizing an object oriented Blackboard architecture, where the various participants involved in the engineering process communicate through a global database - called Blackboard. We have demonstrated the DICE framework through a simulation of the Hyatt

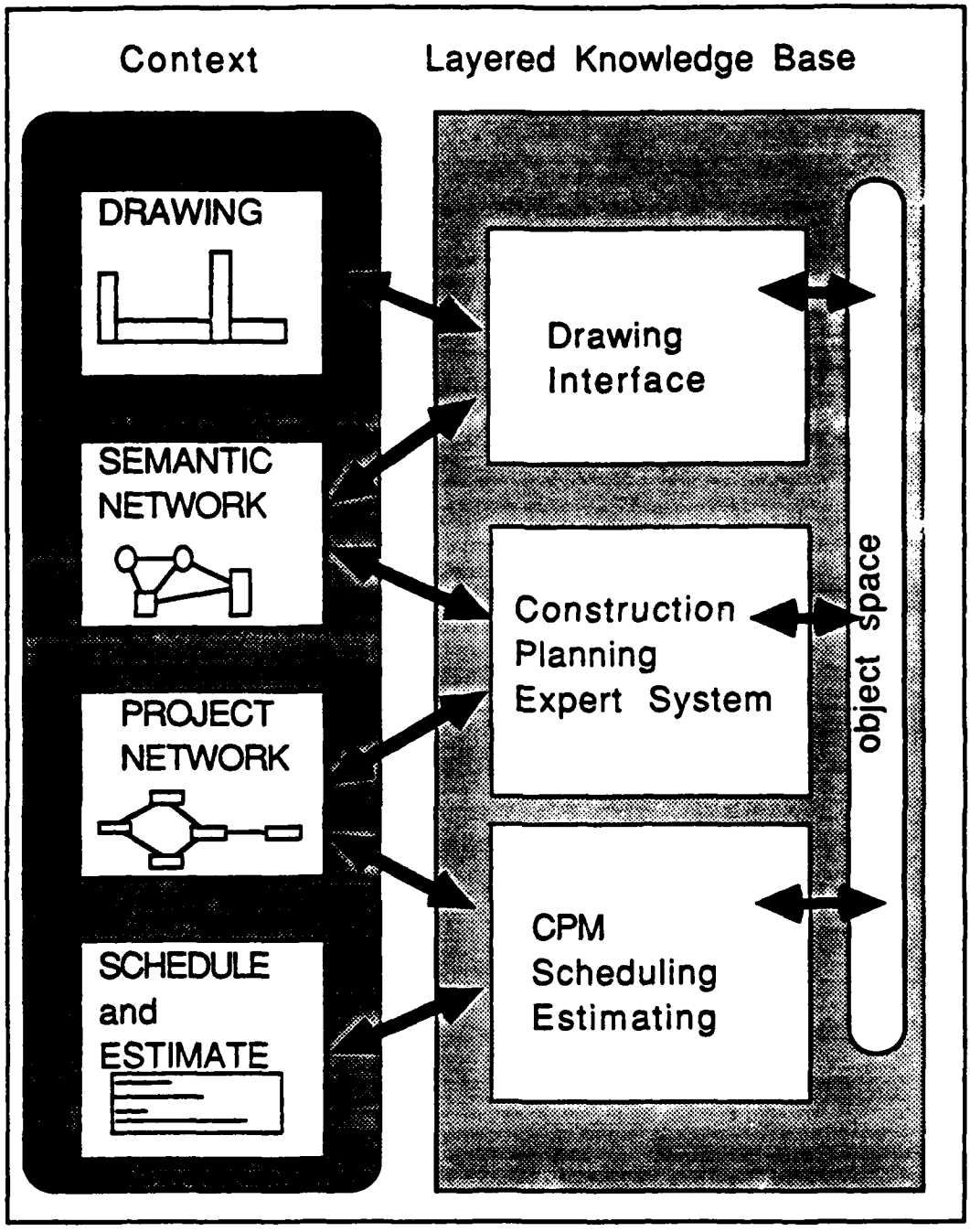


Figure 25: Schematic View of BUILDER

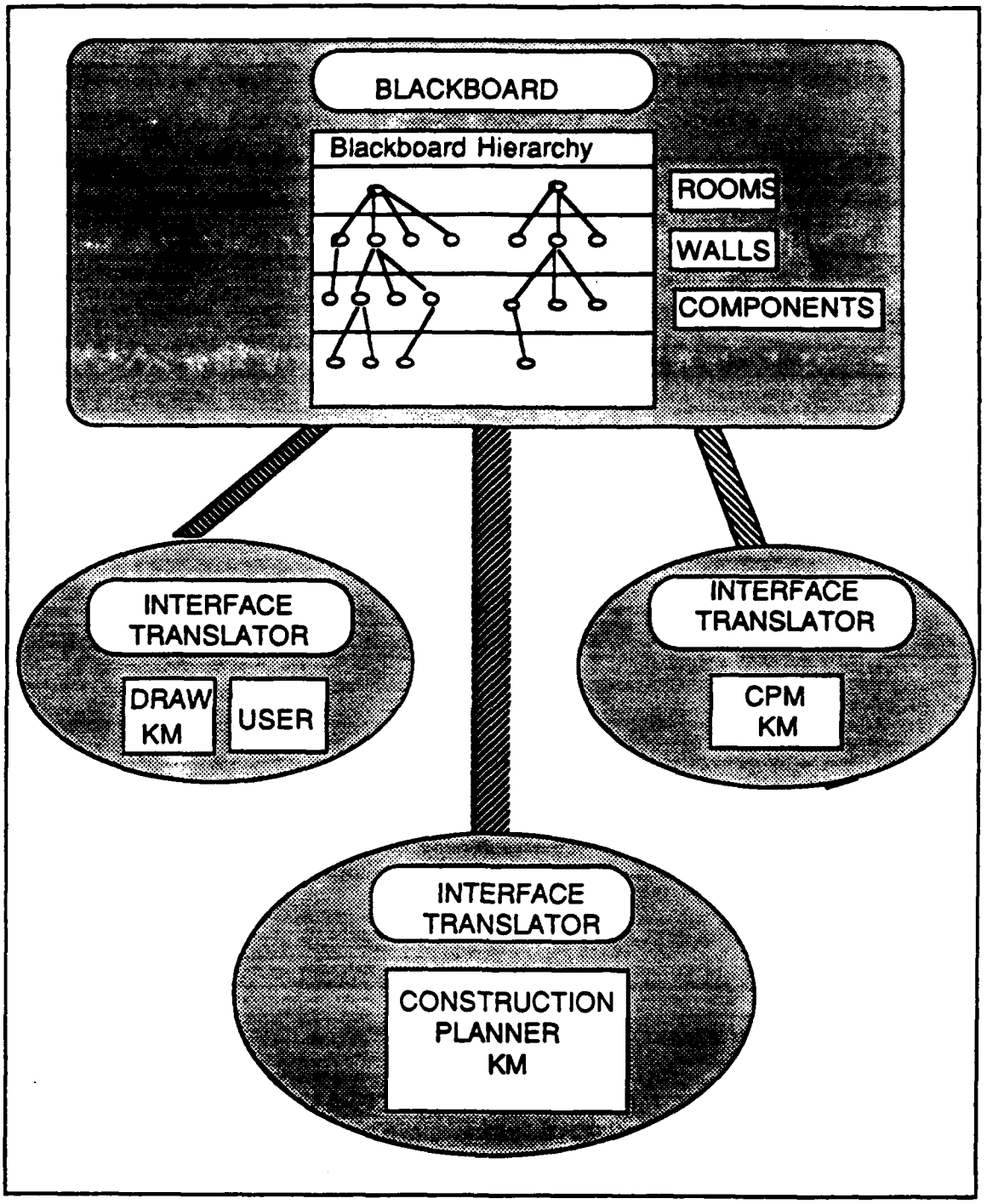


Figure 26: Overview of DICEY-BUILDER

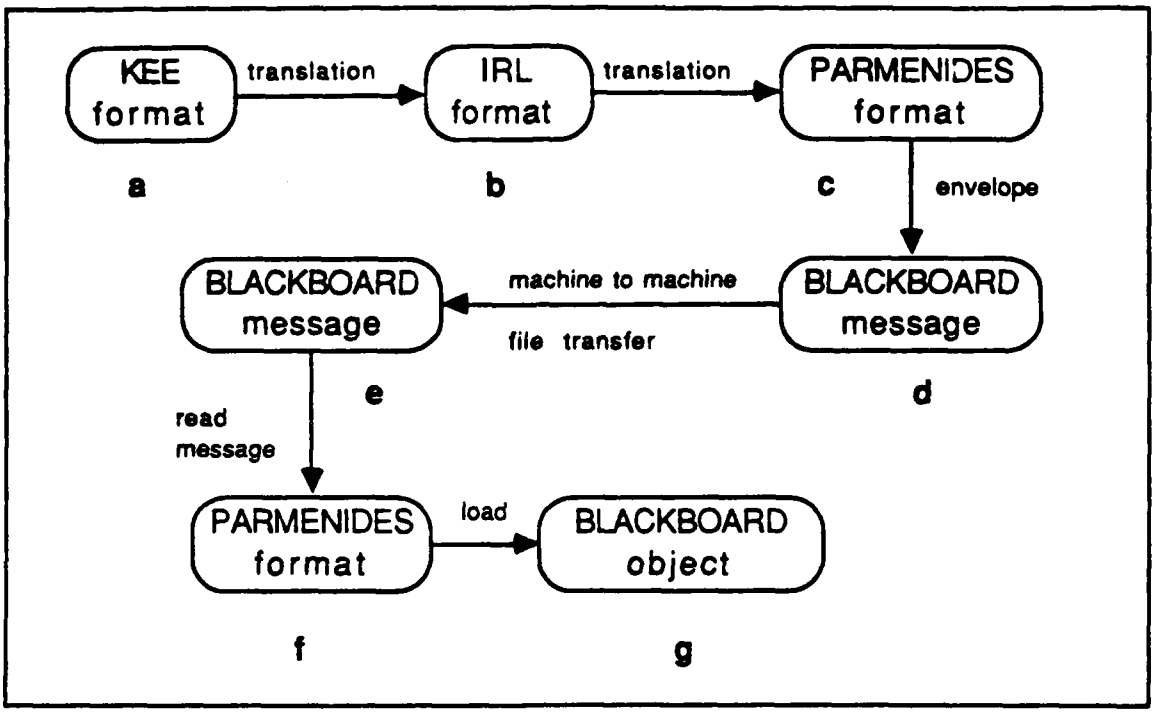


Figure 27: Posting From KM to Blackboard: Translation Process

Regency Disaster and an implementation of a construction planning KBES. Our current efforts are being focused in the following areas: 1) a theory of constraint negotiation for resolving conflicts between various designers, 2) a layered communication protocol that will facilitate an effective communication process between participants in different engineering disciplines, 3) an user interface in the X window system, 4) effective secondary storage management facilities, and 5) demonstrate DICE in an industrial setting.

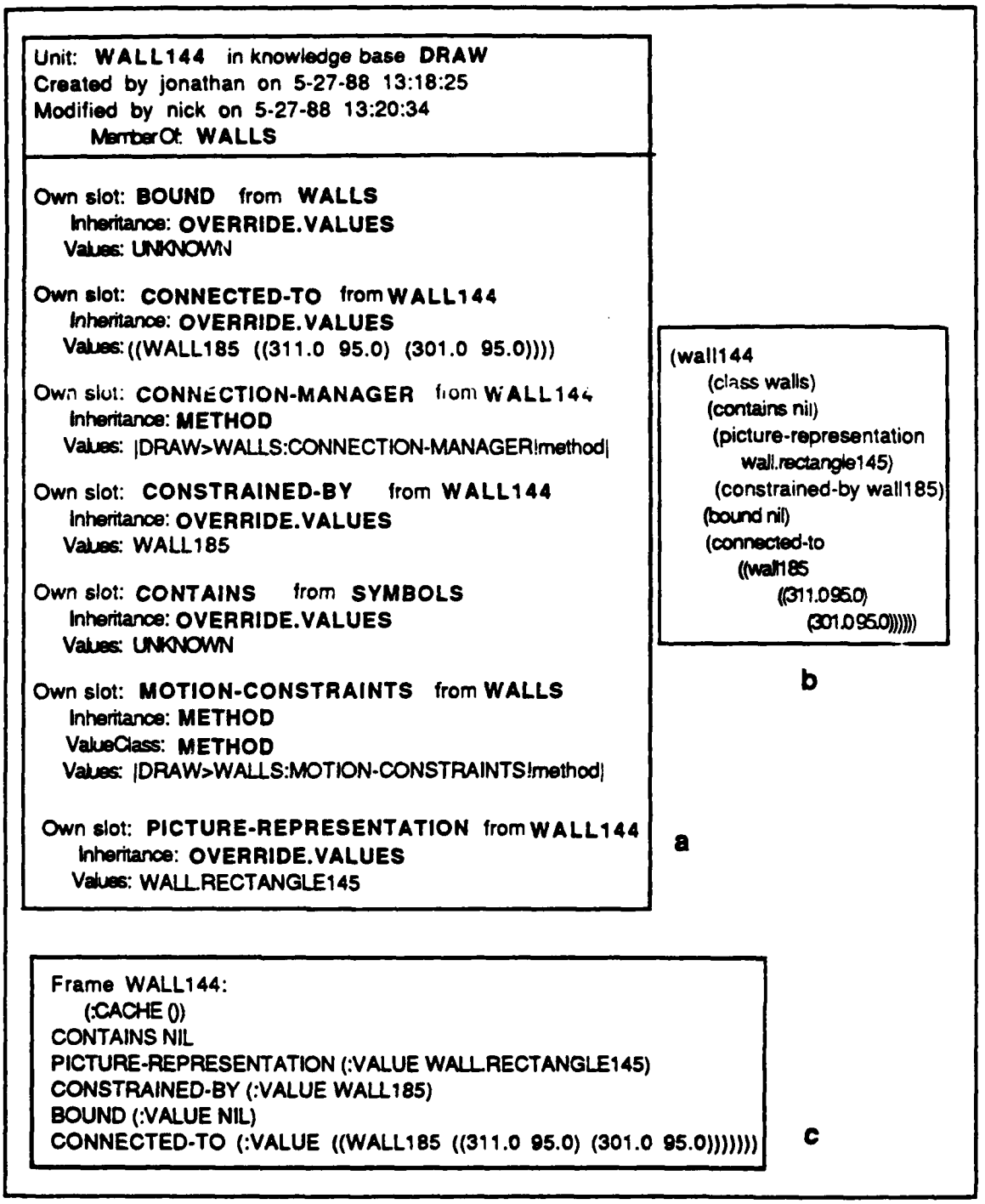


Figure 28: Representative Objects

7 Bibliography

[1] Amarel, S., "Basic Themes and Problems in Current AI Research," *Proceedings of the Fourth Annual AIM Workshop*, Ceilsielske, V. B., Ed., Rutgers University, pp. 28-46, June 1978.

[2] Barton, P. K., *Building Services Integration*, E. F. N. Spon, 733 Third Ave., NY 10017, 1983.

[3] Cellary, W., Gelenbe, E., and Morzy, T., *Concurrency Control in Distributed Database Systems*, Elsevier Science Publishers B.V., 52 Vanderbilt Avenue, NY 10017, 1988.

[4] Chang, S-K., Khikawa, T., and Ligomenides, P. A., Eds., *Visual Languages*, Plenum Press, New York, 1986.

[5] Cherneff, J., *Automatic Generation of Construction Schedules from Arcitectural Drawings*, unpublished Master's Thesis, Dept. of Civil Engineering, M.I.T., Cambridge, MA 02139, 1988.

[6] Conry, S., Meyer, R., Lesser, V., "Multistage Negotiation in Distributed Planning," *Readings in Distributed Artificial intelligence*, pp. pg 367-384, 1988.

[7] Coulouris, G. and Dollimore, J., *Distributed Systems: Concepts and Design*, Addison Wesley, 1988

[8] Davis, R., Smith, R., "Negotiation as a Metaphor for Distributed Problem Solving," *Artificial Intelligence*, Vol. 20, pp. pg 63-109, 1983.

[9] de Kleer, J., "Choices Without Backtracking," *Proceedings of the 4th NCAI*, Texas, AAAI, August 1984.

[10] Doyle, J., "A Truth Maintenance System," *Artificial Intelligence*, Vol. 12, pp. 231-272, 1979.

[11] Eastman, C. M., "Database Facilities for Engineering Design," *Proceedings of the IEEE*, Vol. 69, No. 10, pp. 1249-1263, October 1981.

[12] Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R., "The Hearsay-II Speach Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, Vol. 12, No. 2, pp. 213-253, 1980.

[13] Fenves, S., Flemming, U., Hendrickson, C., Maher, M., Schmitt, G., "An Integrated Software Environment for Building Design and Construction," *Computing in Civil Engineering: Microcomputers to Supercomputers*, 1988.

[14] Fikes, R. and Kehler, T., "The Role of Frame-based Representation in Reasoning," *Communications of the ACM*, pp. 904-920, September 1988.

[15] Groleau, N., *A Blackboard Architecture for Communication And Coordination in Design*, unpublished Master's Thesis, Dept. of Civil Engineering, M.I.T., Cambridge, MA 02139, 1989.

[16] Hayes-Roth, B., "A Blackboard Architecture for Control," *Artificial Intelligence*, Vol. 26, No. 3, pp. 251-321, 1985, [Also Technical Report No. HPP 83-38, Stanford University.].

[17] Howard, H.C., *Integrating Knowledge-Based Systems with Database Management Systems for Structural Engineering Applications*, unpublished Ph.D. Dissertation, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, 1986, [See also Special Issue of CAD, November 1985.].

[18] Katz, R., *Information Management for Engineering Design*, Springer-Verlag , 1985.

[19] "KEE Software Development Manual," Intellicorp, Palo Alto, 1987.

[20] Klein, M., Lu, L., "Run-Time Conflict Resolution in Cooperative Design," *Workshop on AI in Engineering Design, AAAI-88*, 1988.

[21] Krasner, H., "CSCW'86 Conference Summary Report," *AI Magazine*, pp. 87-88, Fall 1987.

[22] Lander, S., Lesser, V., "Negotiation of Conflicts Among design Experts," *Workshop on AI in Engineering Design, AAAI-88*, 1988.

[23] Maier, D., Stein, J., Otis, A., and Purdy, A., "Development of an Object-Oriented DBMS," *OOPSLA 86*, , pp. , 1986.

[24] Marshall, R. D., et al., *Investigation of the Kansas City Hyatt Regency Walkways Collapse*, Technical Report Science Series 143, National Bureau of Standards, Washintong, D. C., May 1982.

[25] Nii, P. and Brown, H., "Blackboard Architectures," AAAI Tutorial Notes No. HA2, 1987.

[26] Nii, P., Fiegenbaum E., Anton, J. and Rockmore, A.. "Signal-to-Symbol Transformation: HASP/SIAP Case Study," *The AI Magazine*, Vol. 3, No. 2, pp. 23-35, Spring 1982.

[27] Papadimitriou, C., *The Theory of Database Concurrency Control*, Computer Science Press, Inc., 1803 Research Boulevard, Rockville, Maryland 20850, 1986.

[28] Pruitt, D.,, *Negotiation Behavior*, Academic Press, 1981.

[29] Reddy, R., Erkes, J., Wood, R., "An Overview of DICE Architecture," *DARPA draft proposal by West Virginia University*, 1988.

[30] Rehak, D.R., Howard. H.C., and Sriram, D., "Architecture of an Integrated Knowledge Based Environment for Structural Engineering Applications," in *Knowledge Engineering in Computer-Aided Design*, Gero, J., Ed., North Holland, 1985.

[31] Rosenschein, J., Genesereth, M., "Deals Among Rational Agents," *Readings in Distributed Artificial Intelligence*, pp. pg 227-234, 1988.

[32] Sathi, A., *Cooperation Through Constraint Directed Negotiation: Study of Resource Reallocation Problems*, Technical Report, Carnegie-Mellon University, Robotics Institute, 1988.

[33] Sathi, A., Morton, T., and Roth, S., "Callisto: An Intelligent Project Management System," *AI Magazine*, Vol. , pp. 34-52, Winter 1986.

[34] Sriram, D., "DESTINY: A Model for Integrated Structural Design," International Journal for AI in Engineering, October, 1986.

[35] Sriram, D., *Knowledge-Based Approaches for Structural Design*, CM Publications, UK, 1987.

[36] Sriram, D., Ed., *Computer Aided Engineering: The Knowledge Frontier*, IESL, Dept. of Civil Engineering, M.I.T., 1989.

[37] Stefik, M. and Bobrow, D., "Object-Oriented Programming: Themes and Variations," *The AI Magazine*, Vol. Winter, pp. 40-62, 1985.